Кафедра радиоэлектроники и защиты информации (РЗИ)

В. В. Шейда

Защита информации в компьютерных сетях. Web уязвимости.

Учебно-методическое пособие для проведения лабораторных работ по дисциплине

«ЗАЩИТА ИНФОРМАЦИОННЫХ ПРОЦЕССОВ В КОМПЬЮТЕРНЫХ СИСТЕМАХ»

для студентов специальности

090103 «Организация и технология защиты информации»
090104 «Комплексная защита объектов информатизации»

2012

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Томский государственный университет систем управления и радиоэлектроники» (ТУСУР)

Кафедра радиоэлектроники и защиты информации
(РЗИ)

УТВЕРЖДАЮ
Зав. каф. РЗИ
_____ А.С.Задорин
«_____»_____ 2012 г.

Защита информации в компьютерных сетях. Web уязвимости.

Учебно-методическое пособие для проведения лабораторных работ по дисциплине «Защита информационных процессов в компьютерных системах»,

для студентов специальностей
090103 «Организация и технология защиты информации»
090104 «Комплексная защита объектов информатизации»

Разработчик:
аспирант кафедры РЗИ
_____ В. В. Шейда

2012

УДК
681.3.067

Рецензент:
профессор каф. РЗИ Задорин А.С.

В.В. Шейда
Защита информации в компьютерных сетях. Web уязвимости. Учебно-методическое пособие для проведения лабораторных работ. / В.В. Шейда. – Томск: ТУСУР, 2012. –68 с

В учебно-методическом пособии кратко рассмотрены основы веб-технологии, принципы построения универсальных идентификаторов ресурсов, взаимодействия веб-сервера и веб-клиента, взаимодействия веб-сервера с модулями обработки программных включений и с базой данных, а также языка JavaScript, позволяющего запускать и выполнять программные включения в веб-страницы в окне браузера на стороне клиента.

Представлены темы лабораторных занятий, выполняемых на учебном веб-сервере, и позволяющие понять причину возникновения уязвимостей в компонентах веб-технологии. В процессе выполнения лабораторных работ, студенты на практике осваивают методы и принципы борьбы против таких уязвимостей.

Методические указания предназначены для студентов очной, заочной и дистанционной форм обучения специальности 090103 и 090104 по дисциплине «Защита информационных процессов в компьютерных системах».

УДК
681.3.067

Содержание

1.1 Унифицированные идентификаторы ресурсов (URI) 6
1.2 Гипертекстовый язык разметки (HTML) 6
2 Протокол передачи гипертекста (HTTP) 7
2.1 HTTP-запрос 7
2.2 Методы протокола HTTP 7
2.3 Заголовки HTTP 9
2.3.1 Заголовки HTTP-запроса 10
2.3.2 Заголовки HTTP ответа 11
2.3.3 Заголовок Connection 12
2.4 Cookies 12
2.4.1. Причины использования cookies 13
2.4.2. Использование cookies в браузере 14
2.4.3 Контроль пользователя над cookies 14
2.4.4 Проблемы нарушения конфиденциальности, связанные с cookies 15
2.4.5 Создание и использование cookies 16
3 Протокол TCP 19
3.1 Установление TCP соединения 20
3.2 Разрыв TCP соединения 21
3.3 Надежность передачи данных 22
4 Обзор языка разметки HTML 24
4.1 Базовые элементы языка и структура HTML документа 24
4.2 Ссылки в HTML документах 25
4.3 Форматирование текста 26
Заголовок 1 уровня 27
Заголовок 2 уровня 27
Заголовок 3 уровня 27
4.4 Форматирование табличных данных 28
5 Обзор языка запросов SQL 30
6 Обзор языка программирования PHP 32
6.1 Переменные в PHP 33
6.2 Управляющие структуры 33
6.3 Основные функции PHP 35
7 Обзор языка программирования JavaScript 40
8 Описание лабораторного web-сайта 42
9 Лабораторная работа №1: «Межсайтовый скриптинг» 47
9.1 Введение 47
9.2 Краткая теория 47
9.3 Содержание работы 48
9.4 Порядок выполнения работы 49
9.5 Контрольные вопросы 50
10 Лабораторная работа №2: «SQL инъекция» 51
10.1 Введение 51
10.2 Краткая теория 51
10.4 Порядок выполнения работы 55
10.5 Контрольные вопросы 56
11 Лабораторная работа №3: «PHP-include» 57
11.1 Введение 57
11.2 Краткая теория 57
11.3 Содержание работы 58
11.4 Порядок выполнения работы 59
11.5 Контрольные вопросы 59

12 Лабораторная работа №4: «Прослушивание трафика (Traffic sniffing)» 60
12.1 Введение 60
12.2 Краткая теория 60
12.3 Содержание работы 61
12.4 Порядок выполнения работы 61
12.5 Контрольные вопросы 61
Литература 62
Модель OSI 63
Краткое описание phpMyAdmin 66
Некоторые команды Linux 68

1 Компоненты Web технологии

В Web имеются четыре семантических компоненты: унифицированные идентификаторы ресурсов (Uniform Resource Identifiers - URI), гипертекстовый язык разметки (Hypertext Markup Language – HTML), протокол передачи гипертекста (Hypertext Transfer Protocol - HTTP) и базы данных. URI представляют собой механизм именования ресурсов в Web с целью их идентификации. HTML – это стандартный язык для создания гипертекстовых документов. HTTP – это протокол для взаимодействия между Web-клиентами и Web-серверами. База данных – хранилище данных, используемых web приложениями. На рисунке 1.1 графически представлены компоненты web.

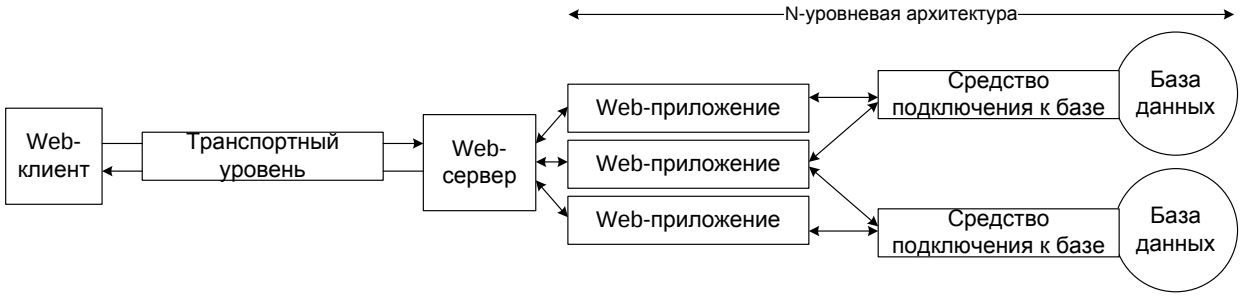

Рисунок 1.1 – Компоненты Web-технологии

1.1 Унифицированные идентификаторы ресурсов (URI)

Доступ к ресурсам Web требует их идентификации. Web-ресурс идентифицируется с помощью унифицированного идентификатора ресурса (URI). Не являясь какой-либо физической сущностью, URI может быть воспринят как некий указатель, к которому применимы методы запросов с целью генерирования потенциально различных ответов в разное время. *Метод запроса* это простая операция, такая как выборка, изменение или удаление ресурса. URI идентифицирует ресурс не зависимо от его текущего местонахождения или содержания. На верхнем уровне URI представляет собой простую форматированную строку, такую как http://www.foo.com/coolpic.gif. URI обычно состоит из трех частей: протокола для взаимодействия с сервером (например, http), имени сервера (например, www.foo.com) и имени ресурса (например, coolpic.gif). Наиболее популярной формой URI является унифицированный указатель ресурса - Uniform Resource Locator (URL).

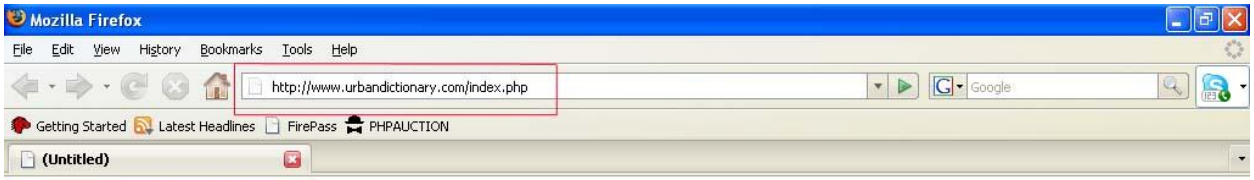

Рисунок 1.2 – пример использование URI

1.2 Гипертекстовый язык разметки (HTML)

Гипертекстовый язык разметки (HTML) обеспечивает стандартное представление гипертекстовых документов (web-страниц) в текстовом формате. HTML позволяет форматировать текст, встраивать в документ изображения, а также создавать гипертекстовые ссылки на другие документы. Синтаксис HTML достаточно прозрачен и прост в изучении. На протяжении ряда лет было разработано множество программных средств для создания и редактирования HTML-файлов, однако для этих целей достаточно и простого текстового редактора, например, Notepad. HTML – документ сам по себе статичен, для генерации динамических HTML-страниц используют языки программирования такие как PHP, PERL и др.

2 Протокол передачи гипертекста (HTTP)

Функционирование Web зависит от наличия стандартного, устоявшегося способа для взаимодействия Web-компонентов. Протокол передачи гипертекста Hypertext Transfer Protocol (HTTP) представляет собой наиболее распространенный способ передачи ресурсов Web. HTTP определяет формат и назначение сообщений, которыми обмениваются Web-компоненты, такие как клиенты и серверы. Протокол – это язык схожий с естественными человеческими языками. Подобно другим языкам, протокол имеет свой собственный синтаксис и семантику, связанные с использованием элементов языка. HTTP определяет синтаксис сообщений и способ интерпретации полей каждой строки сообщения. HTTP представляет собой протокол типа *запрос-ответ* – клиент отправляет сообщение-запрос, а затем сервер откликается сообщением ответом. Клиентские запросы обычно порождаются действиями пользователя, например, щелчком мышью на гиперссылке или вводом URI в адресной строке браузера. HTTP не сохраняет своего состояния – клиенты и серверы трактуют каждый обмен сообщениями независимо от других, и нет необходимости сохранять какое-либо промежуточное состояние между запросами и ответами.

2.1 HTTP-запрос

Пример запроса HTTP приведен на рисунке 1.2.

```
GET /index.php HTTP/1.1
Host: foo.com
Referer: www.google.com
Cookie: Session=123;
```

Рисунок 2.1 – пример HTTP запроса

В первой строке запроса указываются:

- **метод запроса** (в примере выше – GET), сообщающий протоколу HTTP, какое действие необходимо выполнить над ресурсом.
- **унифицированный идентификатор ресурса** (/index.php), указывающий на определенный ресурс сервера
- **версия используемого протокола HTTP** (в данном случае 1.1).

Далее в отдельных строках идут заголовки запроса, такие как Host, указывающий на web-приложение, ресурс которого запрашивается; Referer, информирующий о предыдущем URI с которого был осуществлен переход на запрашиваемый ресурс и др.

2.2 Методы протокола HTTP

Метод GET

Метод GET предполагает получение любой информации, в форме объекта (например, страницы index.html), заданного с помощью URI. Если URI относится к программе (например, PHP скрипту), генерирующей данные, то в результате в виде объекта будут присланы эти данные, а не исходный текст самого скрипта.

Семантика метода меняется на "условный GET", если сообщение-запрос включает в себя поля заголовка If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match или If-Range. Метод условного GET запрашивает, пересылку объекта только при выполнении требований, описанных в соответствующих полях заголовка. Метод условного GET имеет целью уменьшить ненужное использование сети путем разрешения актуализации кэшированых объектов без

посылки множественных запросов или пересылки данных, которые уже имеются у клиента. Семантика метода GET меняется на "частичный GET", если сообщение запроса включает в себя поле заголовка Range. Метод частичного GET ориентирован на уменьшение ненужного сетевого обмена, допуская пересылку лишь части объекта, которая нужна клиенту, и не пересылая уже имеющихся частей.

При передаче пользовательских параметров на сервер последние помещаются в строку URI после знака вопроса. Если параметров несколько, то они отделяются друг от друга символом &:

```
GET /index.php?id=1&user=hack
```

В приведенном примере на сервер передаются два параметра id со значением 1 и user со значением hack.

Метод HEAD

Метод HEAD идентичен GET за исключением того, что сервер не должен присылать тело сообщения. Метаинформация, содержащаяся в заголовках отклика на запрос HEAD должна быть идентичной информации посланной в отклик на запрос GET. Этот метод может использоваться для получения метаинформации об объекте, указанном в запросе, без передачи тела самого объекта. Этот метод часто используется для тестирования гипертекстных связей на корректность, доступность и актуальность.

Отклик на запрос HEAD может кэшироваться в том смысле, что информация, содержащаяся в отклике, может использоваться для актуализации кэшированных ранее объектов данного ресурса. Если новые значения поля указывают на то, что кэшированный объект отличается от текущего объекта (как это индицируется изменением Content-Length, Content-MD5, ETag или Last-Modified), тогда запись в кэше должна рассматриваться как устаревшая.

Метод POST

Так же как и метод GET, метод POST используется для запроса ресурса с сервера. Отличие метода POST от метода GET заключается в том, что пользовательские параметры передаются не в строке URI, а в теле самого запроса после всех используемых заголовков HTTP. Таким образом, с помощью POST на сервер можно передать большие объемы информации, в отличие от метода GET, длина строки запроса которого ограничена.

Метод PUT

Метод PUT требует, чтобы вложенный объект был запомнен с использованием URI. Если URI относится к уже существующему ресурсу, то вложенный объект следует рассматривать как модифицированную версию объекта на исходном сервере. Если URI не указывает на существующий ресурс и запрашивающий агент пользователя может определить этот URI как новый ресурс, исходный сервер может создать ресурс с этим URI. Если новый ресурс создан, исходный сервер должен информировать об этом агента пользователя, послав код отклик 200 (OK) или 204 (No Content - никакого содержимого) и тем самым, объявляя об успешно выполненном запросе. Если ресурс не может быть создан или модифицирован с помощью Request-URI, должен быть послан соответствующий код отклика, который отражает характер проблемы. Получатель объекта не должен игнорировать любой заголовок Content-* (например, Content-Range), который он не понял или не использовал, а должен в таком случае вернуть код отклика 501 (Not Implemented - не использовано).

Если запрос проходит через кэш и Request-URI идентифицирует один или более кэшированных объектов, эти объекты должны рассматриваться как устаревшие. Отклики этого метода не должны кэшироваться.

Фундаментальное отличие между запросами POST и PUT отражается в различных значениях Request-URI. URI в запросе POST идентифицирует ресурс, который будет работать со вложенным объектом. Этот ресурс может быть процессом приемки данных, шлюзом к другому протоколу или отдельным объектом, который воспринимает аннотации. Напротив, URI в

запросах PUT идентифицируют объекты, заключенные в запросе, - агент пользователя знает, какой URI применить и сервер не должен пытаться посылать запрос другому ресурсу. Если сервер хочет, чтобы запрос был направлен другому URI, он должен послать отклик 301 (Moved Permanently). Агент пользователя может принять свое собственное решение относительно того, следует ли переадресовывать запрос.
Один и тот же ресурс может быть идентифицирован многими URI. Например, статья может иметь URI для идентификации "текущей версии", которая отличается от URI, идентифицирующей каждую конкретную версию. В этом случае запрос PUT на общий URI может дать в результате несколько других URI, определенных исходным сервером. HTTP/1.1 не определяет то, как метод PUT воздействует на состояние исходного сервера. \

Метод DELETE

Метод DELETE требует, чтобы исходный сервер уничтожил ресурс, идентифицируемый Request-URI. Этот метод на исходном сервере может быть отвергнут вмешательством человека (или каким-то иным путем). Клиент не может гарантировать, что операция была выполнена, даже если возвращенный статусный код указывает, что операция завершилась успешно. Однако, сервер не должен сообщать об успехе, если за время отклика он не намерен стереть ресурс или переместить его в недоступное место.
Сообщение об успехе должно иметь код 200 (OK), если отклик включает объект, описывающий статус; 202 (Accepted - принято), если операция еще не произведена или 204 (No Content - Никакого содержимого), если отклик OK, но объекта в нем нет.
Если запрос проходит через кэш, а Request-URI идентифицирует один или более кэшированных объектов, эти объекты следует считать устаревшими (stale). Отклики на этот метод не кэшируемы.

Метод TRACE

Метод TRACE используется для того, чтобы запустить удаленный цикл сообщения-запроса на прикладном уровне. Конечный получатель запроса должен отослать полученное сообщение назад клиенту в виде тела объекта (код = 200 (OK)). Конечным получателем является либо исходный сервер, либо первый прокси или шлюз для получения значения Max-Forwards (0) в запросе (см. раздел 13.31). Запрос TRACE не должен включать в себя объект.
TRACE позволяет клиенту видеть, что получено на другом конце цепи запроса и использовать эти данные для тестирования или диагностики. Значение поля заголовка Via представляет особый интерес, так как оно позволяет отследить всю цепочку запроса. Использование поля заголовка Max-Forwards позволяет клиенту ограничить длину цепи запроса, которая полезна для тестирования цепи прокси, переадресующих сообщения по замкнутому кругу.
В случае успеха отклик должен содержать все сообщение-запрос с Content-Type = "message/http". Отклики этого метода не должны кэшироваться.

2.3 Заголовки HTTP

Заголовки играют важную роль в проколе HTTP и являются основным средством для указания способа обработки запроса. Заголовки могут использоваться для предоставления метаданных о ресурсе, таких как его длина, формат кодирования и язык. Заголовки можно считать описателями ответа и запроса. Заголовок ответа может указывать, допустимо ли кэширование ответа, либо каким образом декодировать сообщение для получения исходного содержания (например какой алгоритм сжатия был применен для его преобразования).

HTTP-сообщение может иметь любое число заголовков, отделяемых символами CR и LF.

Заголовок имеет общий синтаксис, в соответствии с которым имя и значение отделяются друг от друга символом двоеточия “:”. Например, чтобы указать время создания сообщения, заголовок Date должен быть включен в сообщение например следующим образом:

Date: Thu, 22 Mar 2007 08:12:31 GMT

“Date” представляет собой поле имени заголовка, а строка “ Thu, 22 Mar 2007 08:12:31 GMT” – поле значения.

2.3.1 Заголовки HTTP-запроса

Заголовок запроса может использоваться клиентом для отправки информации с запросом или задания ограничений при обслуживании запросов сервером. Передаваемая информация может содержать дополнительные сведения о клиенте, например, идентификационные данные пользователя или агента пользователя, либо информацию для авторизации, необходимую, чтобы запрос был обработан исходным сервером. В HTTP/1.0 определены пять заголовков запроса:

• **Authorization**. Заголовок Authorization используется агентом пользователя для включения соответствующих полномочий, необходимых для доступа к ресурсу. Для некоторых ресурсов сервера доступ разрешается только при наличии соответствующих полномочий. Вот пример заголовка Authorization:

```
Authorization:  Basic YXZpYXRpS29IDizM1NA==
```

Здесь Basic (обычная) обозначает схему аутентификации, согласно которой данные представляются в виде идентификатора пользователя и пароля. Строка `YXZpYXRpS29IDizM1NA==` представляет собой кодированные идентификатор пользователя и пароль в формате Base64 [FB96a]. Формат использует простой алгоритм кодирования и декодирования, а закодированные данные не намного длиннее исходных данных. Другие допустимые в HTTP схемы аутентификации предусматривают шифрование па транспортном уровне.

• **From**. Заголовок запроса From дает возможность пользователю включать в свои идентификационные данные адрес электронной почты. Это полезно для клиентских программ, работающих в качестве агентов (например, агент-робот в спайдере), для идентификации пользователя, в интересах которого функционирует программа. Вот пример заголовка From:

From: gorby@moskvax.com

Следует заметить, что использование заголовка From нежелательно, поскольку он нарушает конфиденциальность пользователя, особенно если это делается без его ведома.

• **If-Modified-Since**. Заголовок If-Modified-Since является примером условного заголовка, указывающего, что запрос может быть обработан различными способами в зависимости от значения, заданного в поле заголовка. Если предыдущий ответ от сервера был каптирован клиентом или посредником, значение, заданное в заголовке ответа Last-Modified, используется в последующем запросе GET в качестве аргумента в заголовке If-Modified-Since. Например, для следующего запроса:

```
GET /foo.html HTTP/1.0
If-Modified-Since:  Sun,   21 May 2000 07:00:25 GMT
```

сервер будет сравнивать значение, заданное в заголовке If-Modified-Since, с текущим значением времени последней модификации ресурса. Время последней модификации ресурса может быть доступно на уровне приложения и зависит от типа сервера. Во многих серверах это значение может быть получено с помощью системного вызова операционной системы (например, stat() или fstat() в UNIX). Если ресурс не изменился с указанного времени Sun, 21 May 2000 07:00:25 GMT, сервер просто вернет ответ 304 Not Modified. Для резко меняющихся ресурсов это поможет избежать ненужных пересылок данных. Серверу не нужно повторно генерировать ресурс, и время ожидания на стороне пользователя снижается, поскольку клиент может получить содержимое локально.

• **Referer**. Поле заголовка Referer дает возможность клиенту включать URI ресурса, от которого был получен запрашиваемый URI. Например, предположим, что пользователь посещает Web-страиицу http://www.cnn.com и щелкает на гиперссылке на ресурс http://www.disasterrelief.org/Disasters/world-glance.html на этой странице. Заголовок Referer в

запросе, передаваемом на http://www.disasterrelief.org, будет содержать строку http://www.cnn.com:

```
GET /Disasters/worldglance.html HTTP/1.0
Referer:   http://www.cnn.com
```

Польза от применения ноля Referer состоит в выявлении устаревших гиперссылок. Однако чаще всего оно становится источником нарушения конфиденциальности пользователя. Поле Referer представляет собой заголовок, который может быть использован исходным сервером для отслеживания действий пользователя через журнал регистрации.

Есть и худшие ситуации. Предположим, что с помощью формы, использующей метод GET, передается номер кредитной карты пользователя. Допустим, что сама форма передается безопасным способом (скажем, с применением SSL), а полученная после выполнения запроса с формой страница содержит ссылку на другую страницу, возможно, находящуюся па сервере S. Теперь, если пользователь переходит к этой странице, журнал регистрации сервера S будет содержать запись с полем Referer, содержащим номер кредитной карты пользователя.

Кроме того, сервер может осуществлять проверку поля Referer с целью отклонения запросов на определенные ресурсы, если ссылки па ресурсы находились на страницах, не контролируемых сервером. Например, предположим, что ссылка на встроенное изображение onlymine.gif, изначально принадлежащее ресурсу A, была скопирована в другой документ, B. Теперь, если браузер попытается отобразить документ B, он должен сформировать запрос на ресурс onlymine.gif и включить в запрос заголовок Referer с полем значения, соответствующим ресурсу B. Сервер может отказать в обработке запроса, поскольку поле Referer не содержит A.

• **User-Agent**. Поле User-Agent может быть использовано для включения информации о версии программного обеспечения браузера, версии операционной системы компьютера клиента и, возможно, каких-либо сведений об аппаратной конфигурации. Вот примеры заголовков User-Agent:

```
User-Agent:   Mozilla/4.03 (Macintosh;   I;   68K,   Nav)
User-Agent:  Mozilla/4.04 [en] C-WorldNet (Win95;   I)
```

Эта информация достаточно полезна. Например, по ней можно получить статистику по используемым браузерам. Сервер может отправить альтернативную версию ресурса, если ему известно, что программное обеспечение определенного браузера не сможет отобразить версию, используемую по умолчанию. Сомнительность использования этого заголовка состоит в отслеживании действий пользователя и потенциальном вторжении в его частную жизнь. Например, предположим, что несколько пользователей, применяющих различные версии браузеров в большой системе с разделением времени, посылают запросы на исходный сервер. В этом случае поле User-Agent может быть использовано для выявления пользователя, пославшего определенные запросы.

2.3.2 Заголовки HTTP ответа

Так же как заголовки запроса используются для отправки дополнительной информации о запросах, заголовки ответа применяются для отправки дополнительной информации об ответе и о сервере, создавшем ответ. Синтаксис строки состояния в заголовке строго фиксирован и не дает возможности включения дополнительной информации. Если заголовок ответа не распознан, он считается заголовком содержимого.

В HTTP/1.0 определены три заголовка ответа:

• **Location**. Заголовок Location используется для направления запроса в место расположения ресурса и полезен при переадресации ответов. Заголовок Location имеет следующий синтаксис:

Location: http://www.foo.com/level1/twosdown/Location.html

Поскольку в ответ па запрос могут быть выбраны различные варианты ресурса, заголовок Location предоставляет возможность идентификации местонахождения выбранного

варианта. Если группа ресурсов реплицирована па нескольких зеркальных сайтах, заголовок Location может быть использован для указания на нужный сайт, с которого клиент должен получить ресурс. Если в результате выполнения запроса (например, POST) был создан новый ресурс, заголовок Location идентифицирует созданный ресурс.

• **Server**. Заголовок ответа Server аналогичен заголовку запроса User-Agent Заголовок Server несет информацию о версии программного обеспечения исходного сервера и любую другую информацию, связанную с его конфигурацией. Вот несколько типичных примеров заголовка Server:

Server: Apache/1.2.6 Red Hat
Server: Netscape-Enterprise/3.6.1
Server: Apache/1.x.y mod_perl mod_ssl mod_foo mod_bar

Значение, указанное в заголовке Server, полезно для выявления и решения проблем в ответе, а также для сбора статистической информации, позволяющей определить наиболее популярные версии серверов. Минусом здесь является то, что, данная информация облегчает атаку па сервер. Подобно заголовку User-Agent, заголовок Server не является обязательным и может не включаться в ответы.

• **WWW-Authenticate**. Заголовок WWW-Authenticate используется для указания клиенту, что ресурс требует аутентификации. Клиенту возвращается ответ 401 Unauthorized, и он может повторно обратиться с запросом, указав соответствующие данные в заголовке Authorization. Например, сервер может отправить такой ответ:

WWW-Authenticate: Basic realm="ChaseChem"

и клиенту следует включить в свой запрос необходимую аутентификационную информацию, как разъяснялось ранее в этом разделе.

2.3.3 Заголовок Connection

Некоторые реализации HTTP/1.0 используют заголовок Connection, чтобы оставить соединение открытым и после завершения одиночной транзакции запрос-ответ. Однако в HTTP/1.1 в соответствии с тенденцией к обеспечению управления соединениями и отправителем, и получателем, заголовок Connection может быть использован для указания, что одна из сторон намерена закрыть соединение. Возможно, сервер не хочет поддерживать слишком много открытых долговременных соединений, или отправитель (например, прокси-сервер) решит, что у него больше нет оснований сохранять открытое соединение с данным конкретным сервером. Обе стороны могут включить заголовок Connection: close, чтобы сообщить о намерении закрыть существующее долговременное соединение независимо от намерений другой стороны.

Общий заголовок Connection имеет широкое назначение: перечислять набор заголовков, которые имеют смысл только для текущего соединения транспортного уровня с соседним сервером. Иначе говоря, сервер, получающий заголовок Connection, анализирует этот заголовок и удаляет все заголовки перечисленные в нем. Вариант Connection: close был введен в качестве способа обеспечения совместимости с (HTTP/1.0) формой долговременных соединений. Некоторые реализации HTTP/1.0 поддерживают заголовок Connection: Keep-Alive. Вместо того чтобы изобретать новый заголовок, семантика которого будет пересекаться с семантикой заголовка Connection, механизм Connection в HTTP/1.1 использовал Keep-Alive в качестве одной из многих возможных лексем заголовка Connection. Защита заголовка (т.е. обеспечение того, что данный заголовок не будет пересылаться прокси-серверами далее) также была согласована с существующей лексемой Keep-Alive. Так как по умолчанию в HTTP/1.1 реализуется долговременное соединение, то это выглядит, как если бы заголовок Connection: Keep-Alive включался в каждое сообщение.

2.4 Cookies

HTTP не сохраняет свое состояние, поэтому Web-серверу не нужно хранить какую-либо информацию о прошлых или будущих запросах. Однако на стороне сервера может иметься существенная причина для сохранения информации о состоянии между запросами в ходе сеанса или даже между сеансами. Например, может оказаться необходимым предоставлять доступ к ряду страниц на сервере только определенной группе пользователей. Если ввод идентификационной информации необходим при каждом обращении пользователя к любой из этих страниц, возникает дополнительная нагрузка как на пользователя (ему необходимо вводить эту информацию), так и на сервер (для обработки этой информации). Дополнительные транзакции также снижают пропускную способность сети. Сохранение определенной информации между запросами сокращает непроизводительные затраты для пользователя, сети и сервера. Браузер играет важную роль в предоставлении необходимой информации о состоянии вместе с пользовательским запросом.

Cookies являются одним из средств сохранения состояния HTTP. Cookie — это информация о состоянии, которая передается Web-сервером браузеру и хранится на компьютере пользователя в интересах сервера. Механизм работы с информацией о состоянии HTTP дает возможность клиентам и серверам сохранять информацию за пределами запроса и ответа па него. Cookies впервые были применены Netscape в 1994 г.. Затем организацией Internet Engineering Task Force (IETF) был начат процесс стандартизации. Механизм рабо-ты с состоянием HTTP, формализующий использование cookies, подробно описан в документе RFC 2965, выпущенном в октябре 2000 г. и представляющем собой предложение по стандарту (Proposed Standard) IETF. RFC 2965 отражает опыт различных реализаций.

В этом разделе мы сначала остановимся па причинах использования cookies. Далее будет исследован механизм использования cookies в браузере. Cookies имеют весьма важное значение в контексте обеспечения конфиденциальности пользователя. В этой связи в научном сообществе имеются различные мнения. Раздел заканчивается рассмотрением различных проблем нарушения конфиденциальности, связанных с cookies.

2.4.1. Причины использования cookies

Сервер передает cookie браузеру вместе со своим ответом, требуя от браузера включать cookie в последующие запросы к серверу. Когда пользователь в следующий раз посещает Web-сайт, браузер включает информацию из cookie в заголовок запроса. Таким образом, Web-сервер способен различать пользователей в ходе сеанса и между различными сеансами. Информация, отправленная в cookie, может быть уникальной для каждого посетителя Web-сайта, что создает условие для индивидуального обслуживания пользователей.

Многие Web-сайты (например, The New York Times, http://www.nytimes.com) требуют, чтобы cookies использовались браузером при загрузке страниц. Сайты могут требовать, чтобы пользователи идентифицировали себя именами и паролями. Если это необходимо делать при каждой загрузке страниц с этого сайта, пользователь вряд ли захочет работать с таким сайтом. Вместо этого необходимая идентификационная информация может передаваться автоматически через cookie.

Типичный пример использования cookie — «магазинная тележка», в которую пользователь помещает купленные им товары. Имеются Web-сайты, на которых пользователи могут выбирать товары, которые они планируют купить, например, книги или компакт-диски. По мере выбора пользователем товаров виртуальная магазинная тележка содержит множество выбранных на данный момент товаров.

Без cookies Web-серверу пришлось бы сохранять состояние всех своих пользователей (их может быть тысячи) на своем компьютере в течение длительного периода времени. Содержимое магазинной тележки или перечень приобретенных в последнее время товаров являются примерами информации о состоянии. В других случаях может сохраняться более подробная информация, например, все купленные пользователем товары или страницы, которые

пользователь посетил в ходе последнего сеанса. Реальное состояние не обязательно сохраняется в cookies полностью; cookies часто используются в качестве индекса для базы данных, в которой Web-сервером сохраняется информация о состоянии пользователя. Сохранение информации о пользователе дает возможность приложению совместно использовать ее с другими приложениями, в том числе с другими серверами. Подобное совместное использование информации без ведома пользователя ведет к угрозе конфиденциальности пользователя.

2.4.2. Использование cookies в браузере

На рис. 2.4 представлен клиент, отправляющий запрос исходному серверу A (этап 1). Исходный сервер в ответ включает заголовок **(Set-Cookie)** со значением cookie **(XYZ)** (этап 2). Во все последующие запросы к исходному серверу A клиент включает cookie (этап 3, передача **Cookie** с запросом в заголовке).

Заметим, что клиент не интерпретирует строку cookie (XYZ) при ее сохранении и включении в последующие запросы. Сервер свободен в построении строки, предназначенной клиенту, сформировавшему запрос, и в изменении механизма построения строки cookie. На рисунке также показано, что обмен cookies фактически осуществляется без ведома пользователя, если только пользователь не потребовал уведомлять его каждый раз при передаче cookie. Такое уведомление мешает работе пользователя, поэтому лишь малая часть пользователей, осведомленных об этой возможности, применяет ее на практике.

Cookies первоначально хранятся в оперативной памяти компьютера пользователя и записываются на долговременное устройство хранения (файлы на диске) при выходе из браузера. В соответствии с указаниями по реализации агенты пользователя могут хранить часто используемые cookies столько, сколько нужно, однако существует ряд ограничений. Cookie может иметь размер до 4 Кбайтов. Браузеры дают возможность сохранять максимум 20 cookies на сервер (или домен) и не более 300 cookies, чтобы избежать перегрузки.

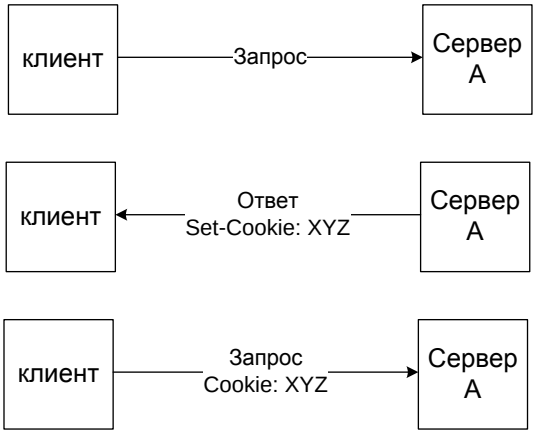

Рис. 2.2 – Обмен cookies между клиентом и сервером

2.4.3 Контроль пользователя над cookies

Как и другие семантические свойства, управление которыми может осуществляться в браузере (см. раздел 2.4.2), cookies допускают значительную степень контроля со стороны пользователя. Пользователи могут:

- **Решать, принимать ли какие-либо cookies вообще.** Подобное решение может сделать для пользователя невозможным загрузку страниц с некоторых сайтов
- **Устанавливать ограничения на размер и количество принимаемых cookie?** Тем самым пользователь управляет пространством, которое будет выделен для cookies на его компьютере, и уменьшает вероятность размещения произвольно больших cookies.

- **Решать, принимать ли cookies от всех сайтов, или только от определенны сайтов/доменов:** Это дает возможность пользователю принимать cookie с нужных сайтов и отклонять cookies с других сайтов.
- **Ограничить время жизни cookies только данным сеансом.** Более детально управление на уровне сеанса дает возможность пользователю разрешать прием cookies только для выполнения определенной задачи. В конце сеанса осуществляется возврат к режиму по умолчанию, в соответствии с которым, cookies для последующих сеансов не принимаются.
- **Потребовать, чтобы cookies исходили от того же сервера, с которого был, получена текущая страница.** Тем самым гарантируется, что пользователь будет знать, откуда поступили cookies, а другим сайтам, с которыми браузер мо автоматически связаться, будет запрещено посылать cookies. Например, когда браузер загружает контейнерный документ, встроенные изображения могу извлекаться автоматически. Встроенные изображения могут размещаться на сервере отличном от того, на котором находится контейнерный документ.

2.4.4 Проблемы нарушения конфиденциальности, связанные с cookies

Пожалуй, ни одна другая технология не вызывала столько разногласий, сколько cookies. Cookies широко используются, но в то же время считается, что они могу привести к нарушению конфиденциальности пользователя. Прежде всего, если применение cookies разрешено браузером по умолчанию, многие пользователи могут даже не подозревать, что они используют cookies. Cookies передаются открытым текстом — другими словами, в незашифрованном виде. «Подслушивающая программа, способная перехватывать сетевой трафик, может узнать содержим» cookies. Создается возможность модификации информации cookies по мере прохождения пакетов по сети. Таким образом, конфиденциальную информацию не следует передавать открытым текстом в составе cookies, если только между клиентом и сервером не установлено защищенное коммуникационное взаимодействие. Помимо раскрытия личной информации пользователя, модификация значений cookies может также привести к изменениям на стороне сервера. Исходный сервер может использовать фрагменты cookies в качестве индексов для обращения к внутренней базе данных, и изменение cookies может привести к непредумышленной модификации базы данных без вмешательства со стороны пользователя. Многие пользователи не знают, кто имеет доступ к информации, извлеченной из cookies, и для чего используется эта информация. Например, информация из cookies может совместно использоваться компаниями, а относящиеся к пользователю данные (профили) могут быть незаконно использованы.

Проблема еще больше усугубляется, если информация из cookies передается серверу, отличному от исходного сервера, которому изначально был послан запрос. Сбор информации третьей стороной может привести к сложно обнаруживаемым и серьезным проблемам с пользовательской безопасностью. Например, агент пользователя U загружает страницу со встроенными изображениями с сервера S. Если встроенные изображения находятся на другом сервере, скажем E, браузер посылает запросы па сервер E, который может передать cookies. Теперь браузер будет иметь cookies и от S, и от E, хотя пользователь явно не обращался к серверу E. Механизм управления промежуточным состоянием HTTP требует, чтобы агент пользователя прекратил прием и отправку cookies в случае не поддающейся проверке транзакции, т.е. пользователю не предоставляется возможность просмотра URL, запрошенного от его имени. Другими словами, запрос к URL делается без ведома пользователя, и cookies принимаются с сайта, который напрямую пользователем не посещался.

Рассмотрим пример. Пусть компания Doubleclick занимается сбором информации. Предположим, пользователь отправляет запрос на поиск информации в популярную поисковую систему и получает десять гиперссылок на страницы с искомой информацией. URL на эти страницы будут не в виде http://www.foo1.com/foo.html, http://www.foo2.com/bar.html и т.д. Вместо этого они могут иметь следующий вид:

http://ad.doubleclick.net/x26362d2/www.foo1.com/foo.html. Теперь, когда пользователь щелкает мышью па одной из гиперссылок, запрос посылается на сервер ad.doubleclick.net. Сервер ad.doubleclick.net записывает информацию из cookie пользователя. Информация из cookie связывает два компонента: пользователя, отправившего запрос, и посещенный им сайт www.fool.com. Далее запрос переадресовывается на сервер www.fool.com. Переадресация представляет собой вид HTTP-ответа, который инструктирует браузер отправить другой запрос с иным URL. Переадресация запросов осуществляется прозрачно для большинства пользователей. Если различные серверы, которые пользуются услугами Doubleclick, сотрудничают между собой и совместно используют информацию из cookies одного и того же пользователя, они могут составить себе более подробный «портрет» пользователя.

Пользователь не может управлять использованием собранной из cookies информации. Было бы полезным при взаимодействии агента пользователя и сервера уведомлять пользователя и запрашивать у него разрешение перед использованием информации. Подобный подход называется моделью подтвержденного участия (opt-in); т.е. пользователи явным образом соглашаются предоставить информацию о себе. Альтернативой является схема уклонения от участия (opt-out), согласно которой поставщики содержания предоставляют пользователям возможность исключения их личных данных из собранной информации. Несмотря на то, что информация о пользователях не собирается автоматически, большинство пользователей не уклоняются от участия в сборе информации о себе. Причинами здесь являются инертность и отсутствие технической грамотности при слепом следовании инструкциям.

Основная проблема с cookies состоит в том, что многие пользователи просто не имеют понятия, какие действия выполняются от их имени. Даже если им известно о наличии cookies, они могут не знать, что можно запретить работу с cookies или использовать их избирательно. Ряд сайтов предоставляет подробную информацию о проблемах, связанных с cookies, и возможном некорректном использовании cookies.

2.4.5 Создание и использование cookies

Cookies дают возможность сохранять информацию о пользователе в течение определенного времени, о чем шла речь выше. Web-сайты используют cookies для отслеживания пользователей и хранения информации о транзакциях, которые выполняются с помощью многочисленных HTTP-взаимодействий. Cookies обычно создаются, используются и модифицируются сценариями, вызываемыми для динамического создания ответов, а не Web-сервером.

Web-сайт может адаптировать содержание, передаваемое пользователю, обратившемуся с запросом. Например, сайт электронной коммерции может возвратить Web-страницу с персонифицированным приветствием, которое содержит имя пользователя и рекомендации по возможным покупкам. Отслеживание пользователей дает возможность Web-сайту создавать профили отдельных пользователей и накапливать статистику о поведении пользователей при обращениях к сайту, например, среднее время, проведенное пользователем на сайте. Однако HTTP-запрос не предоставляет достаточной информации для идентификации пользователей. Множество пользователей могут просматривать Web-содержимое с одного компьютера или направлять свои запросы через один и тот же прокси-сервер. Кроме того, IP-адрес компьютера может время от времени изменяться, если провайдер Internet динамически предоставляет IP-адрес при соединении пользователя с Internet.

Теоретически сайт может потребовать от пользователя указания уникального имени и, возможно, пароля для каждого запроса, но это будет обременительно для пользователя и вызовет у него раздражение.

Вместо этого можно указать браузеру включать уникальный cookie в каждым HTTP-запрос. Вернувшись к нашему примеру, предположим, что Web-сервер получил HTTP-запрос па ресурс http://www.bar.com/book.cgi?name=Noam+Chomsky. Запрос содержит cookie. Сервер вызывает сценарий /www/book/cgi и передает cookie через переменную окружения

HTTP_COOK1E. Сценарий может использовать cookie для определения, какие рекламные объявления и предложения следует поместить на Web-страницу. Например, предложения могут быть связаны с книгами по темам, со ответствующим предыдущим покупкам данного пользователя. С точки зрения браузера cookie представляет собой произвольную строку символов. Сценарий же, со своей стороны, связывает с этой строкой определенный смысл. В простейшем случае строя представляет собой просто уникальный идентификатор пользователя, такой как имя пользователя или числовой код. Если запрос не содержит cookie, сценарий может создать новый cookie и включить его в заголовок сообщения-ответа

```
Set-Cookie: Customer="user17"; Version="1"; Path="/book"
```

Последующие запросы от этого пользователя будут содержать cookie

```
Cookie: Customer="user17"; Version="1"; Path="/book"
```

Сценарий может использовать cookie как идентификатор пользователя при взаимодействии с внутренней базой данных. Например, сайт электронной коммерции может использовать базу данных, хранящую информацию о последних заказах и текущем содержимом магазинной тележки каждого пользователя. База данных сохраняет свое состояние между последовательными запросами в отличие от Web-сервера, который обрабатывает каждый запрос независимо, вызывая сценарий, взаимодействующий с базой данных.

В других ситуациях cookie может содержать дополнительную информации такую как имя пользователя и цвета, которым он отдает предпочтение. Это полезно для динамического создания Web-содержимого, настраиваемого на основе указанных атрибутов. Например, Web-страница может содержать персональное приветствие, включающее имя пользователя. Cookie может также содержать информацию о предыдущих действиях пользователя по просмотру Web-сайта. Хранение накопленной информации в cookie может избавить от необходимости сохранять информацию о пользователе во внутренней базе данных. Например, в cookie сайта электронной коммерции может храниться текущий список заказанных пользователем товаров. Допустим, на сайте электронной коммерции пользователь заполняет различные формы для заказа книг. Пользователь добавляет книгу в список заказываемых товаров, что заставляет браузер послать HTTP-запрос, содержащий cookie, ассоциированный с этим Web-сайтом. Сценарий создает сообщение-ответ, включающий в себя заголовок

```
Set-Cookie: Order="Chomsky_Bio"; Version="1"; Path="/book"
```

Затем пользователь добавляет в список еще одну книгу, что инициирует HTTP-запрос, включающий в себя

```
Cookie: Customer="user17"; Version="1"; Path="/book" Order="Chomsky_Bio";
Version="1"; Path="/book"
```

Процесс продолжается, если пользователь заказывает другие книги. В этом примере все важные данные включены в cookie. Однако при таком подходе cookie может иметь слишком большие размеры. Браузер может не разрешать хранения cookie, превышающего некоторую максимальную длину. Чтобы избежать создания больших cookies, сценарий может использовать cookie лишь для хранения идентификатора пользователя, а содержимое списка заказов пользователя хранить в базе данных на сервере.

Браузер трактует cookie как строку, передаваемую от имени пользователя. Однако достаточно искушенные пользователи могут совместно использовать, создавать или модифицировать cookies. Предположим, что Web-сайт не позволяет пользователю загрузить определенный ресурс, если HTTP-запрос не содержит cookie. Пользователи могут сформировать свои собственные cookies, чтобы не допустить отслеживания сайтом их запросов. Кроме того, один пользователь может «позаимствовать» допустимый cookie у другого пользователя, модифицировав строку в серверном ответе **Set-Cookie.** Обратившись к предыдущему примеру, можно предположить, что искушенный пользователь отправит HTTP-запрос, содержащий cookie, заменив строку **user17** на **user8.** Здесь **user8** соответствует корректному пользователю. Чтобы устранить подобные проблемы, cookie может включать некоторую зашифрованную информацию, не дающую пользователям возможность создавать корректные cookie от своего имени. Как часть обработки запроса сценарий может осуществлять проверку cookie на корректность. Хотя это предотвращает использование фиктивных cookies,

все еще остается риск, что пользователь воспользуется cookie, принадлежащем кому-либо другому. Чтобы предотвратить подобные действия, для cookie может быть назначено время жизни. По истечении времени жизни cookie пользователю необходимо принять новый cookie для получения доступа к сайту.

3 Протокол TCP

Протокол TCP (Transmission Control Protocol, RFC-793, -1323) осуществляет доставку дейтограмм, называемых сегментами, в виде байтовых потоков с установлением соединения. Протокол TCP применяется в тех случаях, когда требуется гарантированная доставка сообщений. Он использует контрольные суммы пакетов для проверки их целостности и освобождает прикладные процессы от необходимости таймаутов и повторных передач для обеспечения надежности. Для отслеживания подтверждения доставки в TCP реализуется алгоритм "скользящего" окна и пакеты подтверждения доставки ACK. Наиболее типичными прикладными процессами, использующими TCP, являются FTP (File Transfer Protocol - протокол передачи файлов) и telnet. Кроме того, TCP используют системы SMTP, HTTP, X-windows, RCP (remote copy), а также "r"-команды. Прикладные процессы (программы) взаимодействуют с модулем TCP через порты.

Примером прикладного процесса, использующего ветвь TCP, может служить FTP, при этом будет работать стек протоколов ftp/tcp/ip/ethernet. Хотя протоколы UDP и TCP могли бы для сходных задач использовать разные номера портов, обычно этого не происходит. Модули TCP и UDP выполняют функции мультиплексоров/демультиплексоров между прикладными процессами и IP-модулем. При поступлении пакета в модуль IP он будет передан в TCP- или UDP-модуль согласно коду, записанному в поле «тип протокола» данного IP-пакета. Формат сегмента (пакета) TCP представлен ниже на рисунке 3.1.

Если IP-протокол работает с адресами, то TCP с портами. Именно с номеров портов отправителя и получателя начинается заголовок TCP-сегмента. Поле код позиции в сообщении определяет порядковый номер первого октета в поле данных пользователя. При значении флага syn=1 в этом поле лежит код isn (см. раздел 3.1 – Установление TCP соединения). Поле HLEN – определяет длину заголовка сегмента, которая измеряется в 32-разрядных словах. Далее следует поле резерв, предназначенное для будущего использования, в настоящее время должно обнуляться. Поле размер окна сообщает, сколько октетов готов принять получатель (флаг ACK=1). Окно имеет принципиальное значение, оно определяет число сегментов, которые могут быть посланы без получения подтверждения. Значение ширины окна может варьироваться во время сессии. Значение этого поля равное нулю также допустимо и указывает, что байты вплоть до указанного в поле номер октета, который должен прийти следующим, получены, но адресат временно не может принимать данные. Разрешение на посылку новой информации может быть дано с помощью посылки сегмента с тем же значением поля номер октета, который должен прийти следующим, но ненулевым значением поля ширины окна. Поле контрольная сумма предназначено для обеспечения целостности сообщения. Контрольное суммирование производится по модулю 1. Перед контрольным суммированием к TCP-сегменту добавляется псевдозаголовок, который включает в себя адреса отправителя и получателя, код протокола и длину сегмента, исключая псевдозаголовок. Поле указатель важной информации представляет собой указатель последнего байта, содержащий информацию, которая требует немедленного реагирования. Поле имеет смысл лишь при флаге URG=1, отмечающем сегмент с первым байтом "важной информации".Если флаг ACK=0, значение поля номер октета, который должен прийти следующим, игнорируется. Флаг urg=1 в случае нажатия пользователем клавиш Del или Ctrl-c.

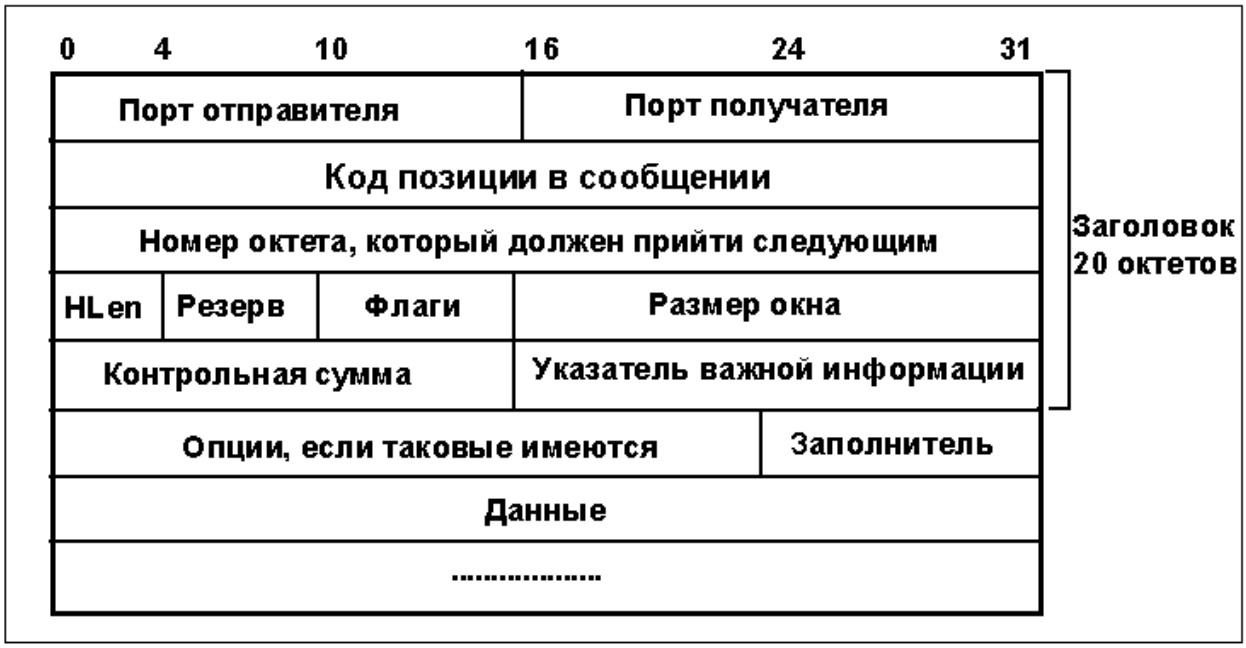

Рисунок 3.1 – Структура заголовка TCP-пакета

В таблице 3.1 приведены возможные значения поля “флаги” заголовка TCP.

Таблица 3.1 – Флаги TCP

urg	Флаг важной информации, поле *Указатель важной информации* имеет смысл, если urg=1.
ack	Подтверждение доставки предыдущего пакета.
psh	Сигнализирует о том, что получатель должен передать данные прикладной программе как можно быстрее.
rst	Прерывание связи.
syn	Флаг для синхронизации номеров сегментов, используется при установлении связи.
fin	Отправитель закончил посылку байтов.

3.1 Установление TCP соединения

Перед тем как TCP протокол будет готов для передачи данных, необходимо установить TCP соединение. Процедура установления TCP соединения (рисунок 3.1) состоит из трех этапов (в литературе часто встречается термин трехфазовое рукопожатие):

1. Запрашивающая сторона (которая, как правило, называется клиент) отправляет SYN сегмент, указывая номер порта сервера, к которому клиент хочет подсоединиться, и исходный номер последовательности клиента (в данном примере ISN, 1415531521). Это сегмент номер 1.
2. Сервер отвечает своим сегментом SYN, содержащим исходный номер последовательности сервера (сегмент 2). Сервер также подтверждает приход SYN клиента с использованием ACK (ISN клиента плюс один). На SYN используется один номер последовательности.
3. Клиент должен подтвердить приход SYN от сервера с использованием ACK (ISN сервера плюс один, сегмент 3).

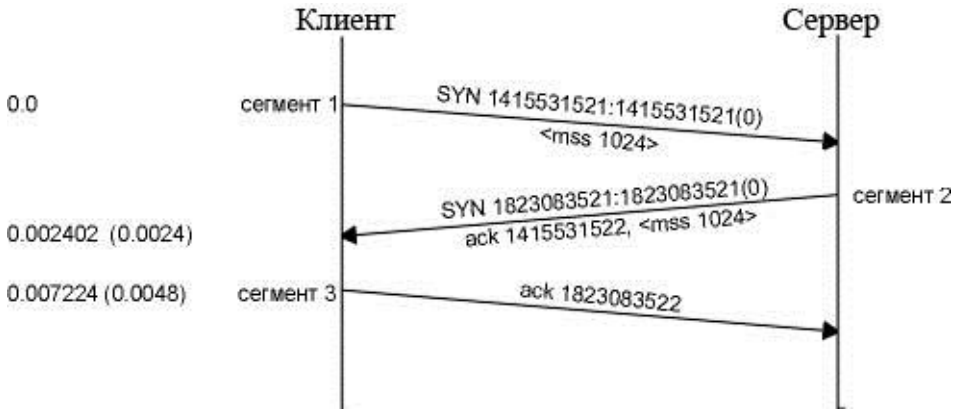

Рисунок 3.1 – Установление TCP соединения

3.2 Разрыв TCP соединения

Для того чтобы установить соединение, необходимо 3 сегмента, а для того чтобы разорвать - 4. Это объясняется тем, что TCP соединение может быть в наполовину закрытом состоянии. Так как TCP соединение полнодуплексное (данные могут передвигаться в каждом направлении независимо от другого направления), каждое направление должно быть закрыто независимо от другого. Правило заключается в том, что каждая сторона должна послать FIN, когда передача данных завершена. Когда TCP принимает FIN, он должен уведомить приложение, что удаленная сторона разрывает соединение и прекращает передачу данных в этом направлении. FIN обычно отправляется в результате того, что приложение было закрыто.

Получение FIN означает только, что в этом направлении прекращается движение потока данных. TCP, получивший FIN, может все еще посылать данные. Несмотря на то, что передача данных при наполовину закрытом TCP соединении возможна, на практике только некоторые (совсем немного) TCP приложения используют это. Обычным является тот сценарий, который показан на рисунке 3.2.

Когда сервер получает FIN, он отправляет назад ACK с принятым номером последовательности плюс один. На FIN тратится один номер последовательности, так же как на SYN. В этот момент TCP сервер также доставляет приложению признак конца файла (end-of-file) (чтобы выключить сервер). Затем сервер закрывает свое соединение, что заставляет его TCP послать FIN, который клиент должен подтвердить (ACK), увеличив на единицу номер принятой последовательности.

На рисунке 3.2 показан типичный обмен сегментами при закрытии соединения. Номера последовательности опущены. На этом рисунке FIN посылаются из-за того, что приложения закрывают свои соединения, тогда как ACK для этих FIN генерируется автоматически программным обеспечением TCP.

Соединения обычно устанавливаются клиентом, то есть первый SYN двигается от клиента к серверу. Однако любая сторона может активно закрыть соединение (послать первый FIN). Часто, однако, именно клиент определяет, когда соединение должно быть разорвано, так как процесс клиента в основном управляется пользователем, который и решает разорвать TCP соединение.

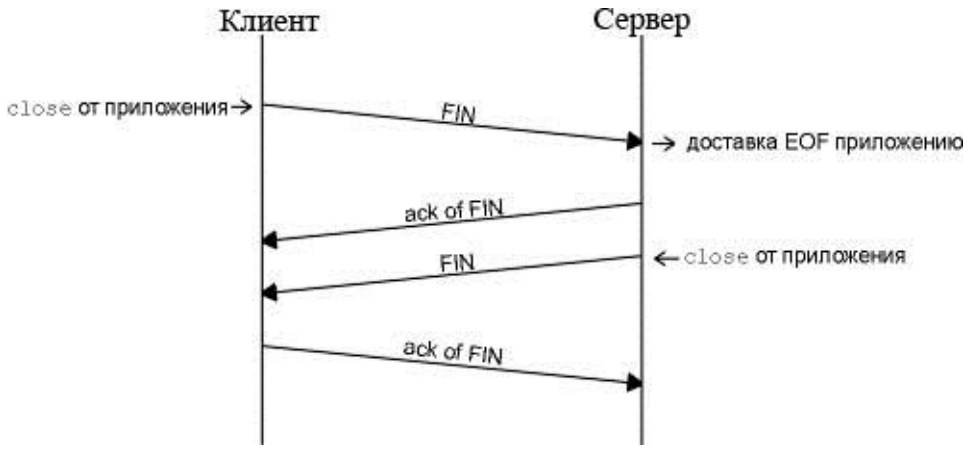

Рисунок 3.2 –Разрыв TCP соединения

3.3 Надежность передачи данных

Надежная сквозная доставка данных является одной из ключевых особенностей TCP. Чтобы обеспечить надежную доставку, протокол TCP должен уметь восстанавливать поврежденные, потерянные, повторяющиеся или доставленные с нарушением исходного порядка данные. Для обеспечения надежности в TCP используется схема повторной передачи с позитивным подтверждением PAR (Positive Acknowledgement Retransmission). На рис. 3.3 показано, как работает схема PAR, когда все данные и подтверждения (пакеты с включенным флагом ACK) поступают без ошибок. Новые данные отправляются лишь после того, как будет подтверждено получение предыдущих данных.

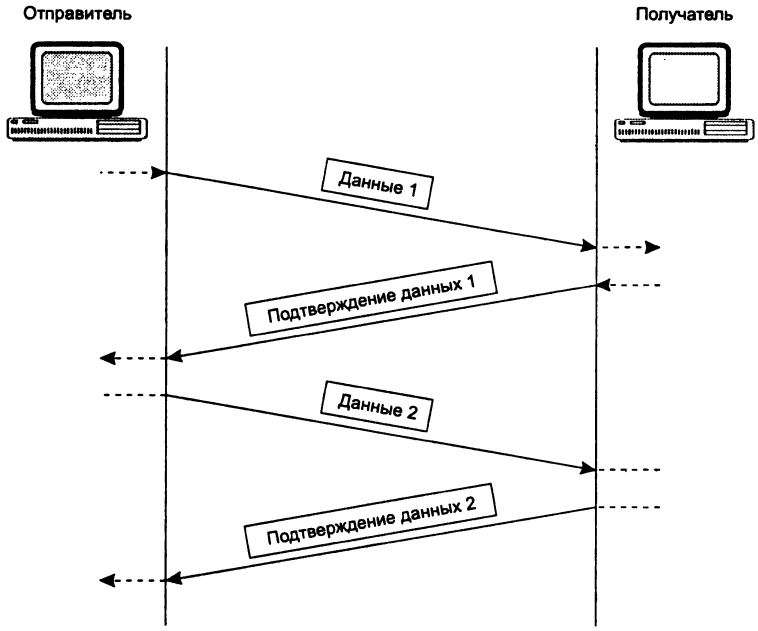

Рисунок 3.3 – Схема PAR при передаче данных без ошибок

На рисунке 3.4 показано, как схема PAR используется при восстановлении потерянных данных.

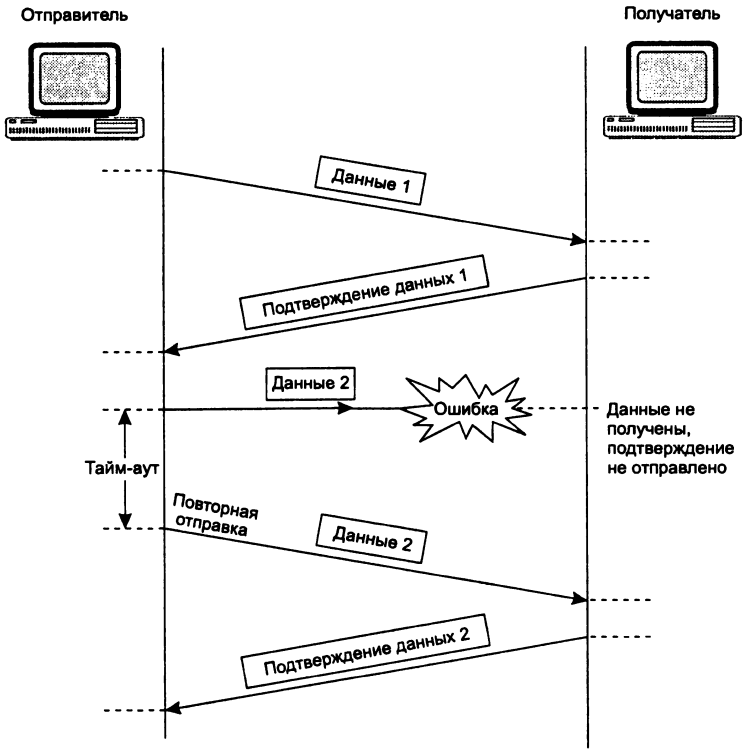

Рисунок 3.4 - Рисунок 3.3 – Схема PAR при восстановлении передачи данных после сбоя

4 Обзор языка разметки HTML

4.1 Базовые элементы языка и структура HTML документа

Базовым элементом языка разметки гипертекста является - *ТЕГ* (*дескриптор, маркер*). Тег всегда заключен между скобками < > и имеет следующий вид:

<ТЕГ атрибут 1=ЗНАЧЕНИЕ ... атрибут N=ЗНАЧЕНИЕ>

Теги бывают одиночными и контейнерными. Контейнером называется пара: открывающий **<ТЕГ>** и закрывающий **</ТЕГ>.**

<ТЕГ> Контейнер </ТЕГ>

Открывающий тег служит для указания программе-браузеру начала какого-либо объекта или задания свойств объектов помещенных в контейнер. Закрывающий тег служит для указания программе-браузеру о конце объекта или окончания применения свойств, заданных в открывающем теге.

Атрибуты тега задают значения свойств данного объекта или объектов помещенных в контейнер. Значения свойств, содержащие пробелы, берутся в кавычки, в остальных случаях кавычки можно опустить.

HTML документ представляет собой обычный текстовый файл, содержащий маркированный тегами форматирования текст, а так же заданные специальными тегами ссылки на графические и прочие файлы мультимедиа, ссылки на другие документы HTML и ресурсы Internet.

Документ HTML начинается открывающим тегом **<HTML>** и заканчивается закрывающим тегом **</HTML>**. Между данной парой контейнерных тегов располагаются две другие основные части HTML документа: заголовок, заключенный в контейнер **<HEAD>...</HEAD>,** и тело документа в контейнере **<BODY>...</BODY>**. Таким образом, структура простого HTML документа выглядит, как показано ниже:

Структура HTML документа

<HTML> - Начало документа

<HEAD>
ЗАГОЛОВОК ДОКУМЕНТА
</HEAD>

<BODY>
ТЕЛО ДОКУМЕНТА
</BODY>

</HTML> - Конец документа

4.2 Ссылки в HTML документах

Как было сказано выше, HTML-документ это обычный текстовый файл. Гипертекстовым его делают содержащиеся в нем ссылки на другие документы и ресурсы Internet. Рассмотрим, что такое ссылка, какие бывают типы ссылок и как их задать в документе.
Ссылка состоит из двух компонентов: так называемого якоря (*anchor*) или элемента привязки и URL (*Universal Resurse Locator*) связанного с ним ресурса Internet.

Первый компонент ссылки - *якорь* это текстовый или графический объект, который, как правило, служит органом управления на Web-странице. Каждый раз при просмотре Web-страниц мы видим множество различных элементов-якорей. Это и красочные рекламные баннеры, всевозможные кнопки и иконки, выделенные подчеркнутым курсивом элементы текста, адреса электронной почты.

Второй компонент ссылки не отображается Web-браузером, но служит конкретным указанием, где в Internet найти, и что сделать при активизации пользователем соответствующего ему якоря.

Адреса ресурсов бывают относительные и абсолютные. Относительный адрес задается в сокращенной форме (путь/файл). Например, если начальная страница index.htm загружена браузером с http://www.site.ru, то использование в ней относительной ссылки resume.htm означает загрузку http://www.site.ru/resume.htm.

Абсолютные адреса используются для привязки к ресурсам других узлов Internet и задаются полным форматом записи (*http://компьютер/путь/файл*). Например: `http://www.sitename.ru/filename.htm`.

Ссылки в документах задаются при помощи контейнера **<A> ...</A>**, следующей структуры:

<A HREF="Ресурс">Элемент - якорь</A>

Атрибут **HREF** в открывающем теге задает ресурс, который необходимо обработать браузеру при выборе на Web-странице, соответствующего ему якоря. Рассмотрим наиболее часто используемые ресурсы:

<A HREF="URL"> - ссылки на другие документы HTML и файлы.
<A HREF="ftp://ftp_server/path/filename"> - ссылки на файлы FTP-сервера.
<A HREF="mailto:e-mail"> - ссылки на адреса электронной почты.
<A HREF="news:newsgroup"> - ссылки на группы новостей.

Заключенный в контейнер элемент-якорь выделяется броузером особым образом (*текст – цветом и подчеркиванием, графику – рамкой*) при отображении на Web-странице.

Рассмотрим несколько конкретных примеров использования ссылок в документах:

<A HREF="http://www.site.ru">Переход на сайт www.site.ru </A> - абсолютная ссылка: переход на сайт `www.site.ru`, текстовый якорь - Переход на сайт www.site.ru

<A HREF="/VW/cars.htm"> Модельный ряд VW </A> - относительная ссылка: открытие станицы cars.htm в подразделе VW относительно раздела основной страницы, текстовый якорь - Модельный ряд VW.

Кроме вышеперечисленных ссылок существуют внутренние ссылки или закладки. Этот тип ссылок используется для удобства перемещения в пределах документа. Для использования

в HTML-документе закладок необходимо задать имена тех областей документа, на которые необходимо ссылаться.

Имя закладки в теле документа задается использованием атрибута **NAME=ИмяЗакладки** в контейнере **<A>...</A>**. Причем в данном случае текст, заключенный в контейнер, не является элементом-якорем (*но выводится*).

Для перехода на начало документа необходимо поместить там закладку:

```
<A NAME=DocBegin>Начало документа<A>
```

Внутренняя ссылка на закладку в документе имеет следующий формат:

<A HREF="URL документа#ИМЯ">Элемент - якорь</A>

Для размещения в документе ссылки на внутреннюю закладку (*содержащуюся в данном документе*) необходимо применить:

```
<A HREF="#DocBegin">Перейти к началу документа</A>
```

Для размещения в документе ссылки на внешнюю закладку (*например, содержащуюся в файле Doc1.htm*) необходимо применить:

```
<A HREF="Doc1.htm#DocBegin">Перейти к началу документа Doc1.htm</A>
```

4.3 Форматирование текста

Данные текстового формата являются доминирующими в HTML документах. Для форматирования текстовых данных в HTML применяются следующие теги:

Теги управления абзацами

<P ALIGN=CENTER/LEFT/RIGHT >...</P> - тег нового абзаца, используется в формате одиночного тега или контейнера. При использовании в форме одиночного тега концом абзаца считается начало следующего, т.е. последующий тег **<P>**. Атрибут **ALIGN** задает выравнивание элементов абзаца, значение по умолчанию **LEFT**

<P>...</P> или <P>	Этот абзац выравнивается по левому краю.
<P ALIGN=CENTER>	Этот абзац выравнивается по центру.
<P ALIGN=RIGHT>	Этот абзац выравнивается по правому краю.

Тег переноса

<BR - тег переноса строк в тексте документа. При разрыве строки межстрочный интервал не увеличивается.

** **	Используется для указания места принудительного разрыва. Пример: **<P>ФИО: Иванов С.С.</P>** Будет выглядеть так: **ФИО:** **Иванов С.С.**

Теги выделения структуры документа

<H1>...</H1>, ... ,<H6>...</H6> - контейнерные теги шестиуровневых заголовков документа. Имеют атрибут **ALIGN** (*по умолчанию LEFT*) для выравнивания заголовка.

<H1>...</H1>	**Заголовок 1 уровня**
<H2>...</H2>	**Заголовок 2 уровня**
<H3>...</H3>	**Заголовок 3 уровня**
<H4 ALIGN=LEFT>...</H4>	**Заголовок 4 уровня по левому краю**
<H5 ALIGN=CENTER >...</H5>	***Заголовок 5 уровня по центу***
<H6 ALIGN=RIGHT>...</H6>	**Заголовок 6 уровня по правому краю**

Теги стилистического выделения текста

Данная группа контейнерных тегов применяется для стилистического выделения элементов текста. Допускается любая комбинация ниже перечисленных тегов.

<B>...</B>	**Выделение полужирным шрифтом**
<I>...</I>	*Выделение курсивом*
<TT>...</TT>	`Выделение телетайпным шрифтом`
<U>...</U>	Выделение подчеркиванием
<STRIKE>...</STRIKE>	~~Выделение перечеркиванием~~
^{...}	Шрифт в верхнем индексе
_{...}	Шрифт в нижнем индексе
<SMALL>...</SMALL>	Мелкий шрифт
<BIG>...</BIG>	Крупный шрифт

Дополнительные теги форматирования

<HR> - тег вставки линии-разделителя. Применяется для визуального разделения текста, при помощи горизонтальных линий. При отображении линии-разделителя в документе, до и после нее, браузер добавляет разделение абзацев.

Комментарии и специальные символы

Для добавления комментариев в HTML документы используется контейнер **<!-- ...-->**. Например:

Кроме комментариев в HTML документах можно использовать специальные символы. Для внедрения специального символа в документ применяется конструкция следующего формата: **&ИМЯ СИМВОЛА**. Специальные символы используются для отображения элементов математических символов (**&divide это ÷, &frac34 это ¾**), редких символов национальных алфавитов и общепринятых символов (**&copy это ©, &reg это ®**).

Например, для отображения на Web-странице HTML тегов необходимо использовать символы заменители угловых скобок (**< это &lt**) и (**> это &gt**). Еще один часто используемый при форматировании (*например, создание пустых ячеек в таблицах*) спецсимвол это неразрывный пробел ** **.

Пример форматирования текстовых данных, используя html.

```
<HTML>
<HEAD>
<TITLE>Форматирование текстовых данных</TITLE>
</HEAD>
<BODY BGCOLOR ="WHITE", TEXT="BLACK", LINK="BLUE", ALINK="RED", VLINK="NAVY"
>
<BASEFONT FACE="Times New Roman","Arial" SIZE=4>
<H1 ALIGN=CENTER>SQL-инъекция</H1>
<HR SIZE=5 COLOR=BLACK>
<FONT SIZE=+2><U>SQL Injection</U></FONT>
<P>
<DFN>SQL Инъекция</DFN> — один из распространённых способов взлома программ,
работающих с базами данных. <BR> Атака, использующая данную уязвимость, возможна
из-за некорректной обработки входных данных, используемых в SQL-запросах. SQL
инъекция позволяет атакующему выполнить произвольный запрос к базе данных,
например, прочитать содержимое любых таблиц или удалить все данные.
</P>
<HR SIZE=5 COLOR=BLACK>
<P ALIGN=CENTER>&copy 2001 Вебмастер
<A HREF="mailto:myname@mail.ru">Написать web-мастеру</A>
</BODY>
</HTML>
```

4.4 Форматирование табличных данных

Таблицы являются важнейшим элементом HTML-документов, так как кроме богатых возможностей по представлению структурированных данных они широко применяются как средство для создания "каркасов" Web-страниц.

Таблицы в HTML определяются при помощи контейнера **<TABLE>...</TABLE>**, заключающего в себе другие элементы таблицы: название, заголовки ячеек и их содержимое. Атрибутами контейнера **<TABLE>** задается формат линии-сетки и общие правила

форматирования (*общий формат действуют, если иной формат не задан атрибутами формата конкретных строк и ячеек*).

Выравнивание наименования задается при помощи атрибута **ALIGN**, который может принимать значения **TOP** (*сверху таблицы*) и **BOTTOM** (*снизу таблицы*).

Данные в таблице организованы построчно, каждая новая строка таблицы задается тегом **<TR>...</TR>.** Для каждой строки таблицы при помощи специальных атрибутов тега <**TR**> можно управлять общим форматированием составляющих строку ячеек.

Строки таблицы разбиваются на ячейки при помощи тегов ячеек-заголовков **<TH>...</TH>** и тегов ячеек-данных **<TD>...</TD>**. Как и для строк таблицы при помощи специальных атрибутов тегов <**TD**> и <**TH**> можно управлять форматированием конкретных ячеек таблицы.

Структура таблицы:

<TABLE> - начало контейнера таблицы
<CAPTION> название таблицы **</CAPTION>**
<TR> <TH> 1 заголовок **</TH>:<TH>** N заголовок **</TH> </TR>** - первая строка / шапка
<TR> <TD> ячейка 1/1 **</TD>:<TD>** ячейка N/1 **</TD> </TR>** - 1 строка
<TR> <TD> ячейка 1/i **</TD>:<TD>** ячейка N/i **</TD> </TR>** - i строка
<TR> <TD> ячейка 1/M **</TD>:<TD>** ячейка N/M **</TD> </TR>** - последняя строка
</TABLE> - конец контейнера таблицы.

5 Обзор языка запросов SQL

В большинстве случаев текстовая информация, представленная на web-страницах в Интернете, хранится в базах данных.

База данных представляет собой структурированную совокупность данных. Эти данные могут быть любыми - от простого списка предстоящих покупок до перечня экспонатов картинной галереи или огромного количества информации в корпоративной сети. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, используется язык запросов SQL. В таблице 5.1 кратко описаны основные команды и синтаксис этого языка.

Таблица 5.1 – Основные команды языка SQL

Назначение команды:	**Команда на языке SQL**
Создание новой базы	create database *ИмяБазы*[1];
Вывод списка существующих баз	show databases;
Вывод списка существующих в базе таблиц	show tables;
Вывод характеристик полей базы	describe *ИмяТаблицы*;
Удаление базы	drop database *ИмяБазы*;
Удаление таблицы	drop table *ИмяТаблицы*;
Ввод данных в таблицу	insert into *ИмяТаблицы* (*ИмяЯчейки1, ИмяЯчейки2*, …) values (*‘Значение1’, ‘Значение2’*,…);
Вывод данных хранящихся в таблице	SELECT * FROM *ИмяТаблицы*;
Вывод строк таблицы, в поле [field name] которых, хранится значение whatever	SELECT * FROM *ИмяТаблицы* WHERE *ИмяЯчейки* = " *Значение*";
Вывод строк таблицы users, в полях name и phone_number которых, хранятся значения “Bob” и '3444444' соответственно	SELECT * FROM *users* WHERE *name* = "Bob" AND *phone_number* = '3444444';
Вывод всех записей, в ячейке name которых записи начинаются с ‘bob,’ а ячейка phone_number содержит ‘3444444'	SELECT * FROM *users* WHERE *name* like "Bob%" AND *phone_number* = '3444444';
Вывод количества строк таблицы	SELECT COUNT(*) FROM *ИмяТаблицы*;
Суммирование столбцов	SELECT SUM(*) FROM *ИмяТаблицы*;
Изменение данных, хранящихся в таблице	UPDATE *ИмяТаблицы* SET Select_priv = 'Y', Insert_priv = 'Y', Update_priv = 'Y' where *ИмяЯчейки* = 'user';
Удаление строки, в поле *ИмяЯчейки* которой хранится значение whatever	DELETE from *ИмяТаблицы* where *ИмяЯчейки* = 'whatever';
Удаление столбца	alter table *ИмяТаблицы* drop column *ИмяЯчейки*;
Добавление нового столбца в таблицу	alter table [table name] add column [new column name] varchar (20);
Изменение имени столбца	alter table *ИмяТаблицы* change *СтароеИмяЯчейки НовоеИмяЯчейки* varchar (50);
Изменение размера столбца.	alter table *ИмяТаблицы* modify *ИмяЯчейки* VARCHAR(3);

[1] Вместо *ИмяТаблицы, ИмяЯчейки* в реальных SQL запросах необходимо подставить конкретные имена таблиц и ячеек базы данных

SQL-запросы не могут быть использованы непосредственно с помощью HTML, так как последний – это язык разметки предназначенный для форматирования текстовой информации. Для того чтобы извлечь текстовую информацию из базы данных с помощью SQL-запросов необходимо использовать более функциональный язык программирования, например PHP.

6 Обзор языка программирования PHP

PHP ("PHP: Hypertext Preprocessor") – это широко используемый язык программирования общего назначения с открытым исходным кодом. PHP сконструирован специально для ведения Web-разработок и может внедряться в HTML-код.

```
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>

        <?php
        echo "Привет, я - скрипт PHP!";
        ?>

    </body>
</html>
```

Рисунок 6.1 – Пример программы на PHP встроенной в HTML

Отличие скрипта представленного на рисунке 6.1 от скриптов, написанных на других языках, например, на Perl или C – заключается в том, что вместо того, чтобы создавать программу, которая занимается формированием HTML-кода и содержит множество предназначенных для этого команд, вы создаете HTML-код с несколькими внедренными командами PHP (в приведенном случае, предназначенными для вывода текста). Код PHP отделяется специальными начальным “<?php” и конечным “?>” тегами, которые позволяют процессору PHP определять начало и конец участка HTML-кода, содержащего PHP-скрипт.

Значительным отличием PHP от какого-либо кода, выполняющегося на стороне клиента, например, JavaScript, является то, что PHP-скрипты выполняются на сервере. Если бы на web-сервере был размещен скрипт, подобный приведенному выше (рисунок 6.1), клиент получил бы только результат выполнения скрипта, т.е. страницу html – сам PHP код на странице полученной клиентом не отображается.

PHP крайне прост для освоения, но вместе с тем способен удовлетворить запросы профессиональных программистов. Хотя PHP, главным образом, предназначен для работы в среде web-серверов, область его применения не ограничивается только этим

6.1 Переменные в PHP

Переменные в PHP представлены знаком доллара с последующим именем переменной. Имя переменной чувствительно к регистру.

Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP. Правильное имя переменной должно начинаться с буквы или символа подчеркивания с последующими в любом количестве буквами, цифрами или символами подчеркивания.

PHP поддерживает восемь простых типов.

Четыре скалярных типа:

- boolean
- integer
- float (число с плавающей точкой или 'double')
- string

Два смешанных типа:

- array
- object

И, наконец, два специальных типа:

- resource
- NULL

В процессе программирования на PHP нет необходимости устанавливать тип переменной вручную; предпочтительнее, чтобы это делал PHP во время выполнения программы в зависимости от контекста, в котором используется переменная.

Любому запускаемому скрипту PHP предоставляет большое количество предопределенных переменных, наиболее важные из них: $_GET и $_POST.

$_GET – Массив, содержащий переменные, передаваемые скрипту через HTTP запрос с методом GET (т.е. переменные, передаваемые в строке запроса URI).

$_POST – Массив, содержащий переменные, передаваемые скрипту через HTTP запрос с методом POST (т.е. переменные, передаваемые в теле HTTP запроса, например данные формы).

6.2 Управляющие структуры

Конструкция *if (выражение)…elseif…else* – это обычный оператор ветвления, который в зависимости от того истинно или ложно выражение, заключенное в скобки выполняет ту или иную последовательность операторов.

```
<?php
if ($a > $b) {
    echo "a больше, чем b";
} elseif ($a == $b) {
    echo "a равно b";
} else {
    echo "a меньше b";
}
?>
```

Рисунок 6.2 – Пример использования конструкции *if (выражение)…elseif…else*

Конструкция *while (выражение){оператор}* – это классический цикл с предусловием. Пока выражение в скобках истинно будет выполнятся операторы заключенные в тело цикла.

```
<?php
```

```
$i = 1;
while ($i <= 10) {
    echo $i;
    $i++;
}
?>
```

Рисунок 6.3 – Пример использования конструкции *while (выражение){оператор}*

Цикл *for* – классический цикл со счетчиком. Синтаксис цикла показан на рисунке 6.4

```
<?php

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
?>
```

Рисунок 6.4 – Пример использования цикла *for*

Оператор *switch* позволяет отказаться от использования последовательности операторов *if* примененных к одному выражению. Часто существует необходимость сравнить одну переменную с разными значениями и в зависимости от того, какое значение содержится в переменной выполнить тот или иной код. Именно это и позволяет реализовать оператор *switch.*

```
<?php
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}

switch ($i) {
case 0:
    echo "i equals 0";
    break;
case 1:
    echo "i equals 1";
    break;
case 2:
    echo "i equals 2";
    break;
}
?>
```

Рисунок 6.5 – Пример использование оператора *switch*

6.3 Основные функции PHP

Функции позволяющие подключать дополнительные модули к скриптам PHP

Функции include() и require() позволяют включить файл в код программы. Программа в таком случае будет обработана PHP с учетом включенного в нее файла.

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>

test.php
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

Рисунок 6.6 – Пример использование оператора функции *include()*

Основное отличие функций *include()* и *require()* заключается в обработке нештатных ситуаций. В случае использования *require()* нештатная ситуация (например, отсутствие включаемого файла), приводит к полной остановке выполнения скрипта, тогда как при использовании *include()* программа продолжит выполнение со следующего по порядку оператора.

Функции для работы с регулярными выражениями и строковыми переменными.

preg_replace(*$pattern, $replacement, $string*) – функция, предназначенная для замены текста, хранящегося в переменной *$string* и удовлетворяющего шаблону *$pattern* на строку, хранящуюся в *$replacement.*

```
<?php

$replacement=$content;
$regexp='/пример/';
$html=preg_replace($regexp, $replacement, $tmpl);
echo $html;

?>
```

Рисунок 6.7 – Пример использования функции *preg_replace*

В примере на рисунке 6.6 слово *пример,* хранящееся в переменной *$tmpl,* заменяется содержимым переменной *$replacement.*

preg_match_all($pattern, $string, $matches, PREG_SET_ORDER) - функция, предназначенная для поиска шаблона *$pattern* в строке *$string*. Результат записывается в массив *$matches*. *PREG_SET_ORDER* – флаг, устанавливающий способ формирования массива *$matches*. В данном случае массив будет сформирован следующим образом: $matches[0] – массив, содержащий первую группу совпадений, $matches[1] – 2-ую и т.д.

implode($glue, $array) – функция возвращает строку, которая представляет собой элементы массива $array «склеенные» вместе с помощью *$glue* (рисунок 6.6).

```php
<?php

$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);

echo $comma_separated; // lastname,email,phone

?>
```

Рисунок 6.8 – пример использования функции *implode*

В примере, приведенном на рисунке 6.7, элементы массива *$array* объединяются в строку *$comma_separated*, «склеиваясь» с помощью запятой.

htmlentities ($string, $quote_style,$charset) – функция конвертирует все соответствующие символы в коды HTML. Переменная *$quote_style* определяет, следует ли конвертировать кавычки. Значения, которые может принимать переменная *$quote_style,* представлены в таблице 6.1. Переменная *$charset* задает используемую кодировку, по умолчанию используется ISO-8859-1. Значения, которые может принимать переменная *$charset*, представлены в таблице 6.2.

Таблица 6.1 – Значения переменной *$quote_style*

Значение	**Description**
ENT_COMPAT	Функция конвертирует только двойные кавычки
ENT_QUOTES	Функция конвертирует все типы кавычек
ENT_NOQUOTES	Функция не будет конвертировать кавычки

Таблица 6.2 – Значения переменной *$charset*

Кодировка	**Описание**
ISO-8859-1	Западноевропейская, Латиница-1
ISO-8859-15	Западноевропейская, Латиница -9. Содержит знак евро, французские и финские буквы, которые отсутствуют в Латинице-1(ISO-8859-1).
UTF-8	ASCII совместимая мультибайтная 8-битовая Unicode кодировка.
cp866	Кириллица для DOS
cp1251	Кириллица для Windows
cp1252	Западноевропейская кодировка для Windows.
KOI8-R	Русская кодировка

BIG5	Традиционная китайская кодировка, главным образом использующаяся на Тайване
GB2312	Упрощенная китайская кодировка, национальный стандарт
BIG5-HKSCS	Big5 расширенная для Гонконга, Традиционная китайская
Shift_JIS	Японская
EUC-JP	Японская

Функции для работы с файлами

fopen($path,mode) – функция открывает файл (или другой ресурс) для записи, чтения и т.д., в зависимости от режима *mode*. Список всех режимов представлен в таблице 6.1

Таблица 6.3 – Режимы для работы с файлами в PHP

Режим (mode)	**Описание**
‘r’	Открывает файл только для чтения.
‘r+’	Открывает файл для чтения и записи.
‘w’	Открывает файл для записи. Если файла не существует, то создает его, если файл существует – обнуляет его.
‘w+’	Открывает файл для записи и чтения. Если файла не существует, то создает его, если файл существует – обнуляет его.
‘a’	Открывает файл для добавления информации в него. Новые данные будут записаны в конец файла.
‘a+’	Открывает файл для чтения и добавления информации в него. Новые данные будут записаны в конец файла.

fread($handle,$length) – считывает *$length* байт из файла указываемого переменной *$handle*. Для успешного выполнения данной функции файл должен быть открыт с помощью функции fopen в режиме чтения (‘r’).

```php
<?php
// get contents of a file into a string
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
```

Рисунок 6.9 – Пример использования функций *fopen* и *fread*

fwrite($handle, $content) – записывает данные *$content* в файл указываемый переменной *$handle*. Для успешного выполнения данной функции файл должен быть открыт с помощью функции fopen в режиме записи или добавления (‘w’ или ‘a’).

```php
<?php
$filename = 'test.txt';
$somecontent = "Add this to the file\n";

// Функция is_writable проверяет доступен ли файл для записи.
if (is_writable($filename)) {
```

```
    // В данном примере файл открывается в режиме добавления,
    // поэтому данные будут дописаны к концу файла

    if (!$handle = fopen($filename, 'a')) {
         echo "Cannot open file ($filename)";
         exit;
    }

    // Записываем $somecontent в открытый файл.
    if (fwrite($handle, $somecontent) === FALSE) {
        echo "Cannot write to file ($filename)";
        exit;
    }

    echo "Success, wrote ($somecontent) to file ($filename)";

    fclose($handle);

} else {
    echo "The file $filename is not writable";
}
?>
```

Рисунок 6.10 – Пример использования функций fopen и fwrite

file($path) – считывает весь файл, находящийся по пути *$path*, в массив

```
<?php

$lines=file($file_name); //Считываем файл в массив $lines

$result=implode('',$lines); //Объединяем элементы массива
                            //(по сути строки файла) в строку $result

?>
```

Рисунок 6.11 – Пример использования функциий file и implode

Функции для работы с базой данных mySQL:

mysql_connect($address, $mysql_user, $mysql_password) – устанавливает соединение с сервером mySQL, расположенным по адресу *$address*. Для успешного соединения необходимо указать пользователя базы *$mysql_user* и его пароль *$mysql_password*. Функция возвращает идентификатор соединения.

$mysql_select_db($db_name, $link) – выбирает базу *$db_name*, к которой необходимо подсоединиться, *$link* - идентификатор соединения, возвращенный функцией *mysql_connect*.

mysql_query($query) – посылает запрос *$query* в базу данных. В качестве запроса может быть использована любая команда языка SQL (таблица 3.1). Функция возвращает указатель на данные, полученные с помощью запроса (рисунок 6.10, переменная *$result*).

mysql_fetch_array($result, format) – формирует массив с данными полученными в результате запроса mysql_query. format указывает типа массива: *MYSQL_NUM* – массив с номерованными индексами, *MYSQL_ASSOC* – массив с индексами по имени столбцов таблицы.

mysql_close($link) – функция закрывает соединение с базой данных, где *$link* – идентификатор соединения с базой, тот который был получен в результате выполнения функции *mysql_connect*

```php
<?php


$link=mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb",$link);

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("ID: %s  Name: %s", $row["id"], $row["name"]);
}

mysql_free_result($result);
mysql_close_db($link);

?>
```

Рисунок 6.12 – Пример использования функций *mysql_connect*, *mysql_query*, *mysql_fetch_array* и *mysql_close*

7 Обзор языка программирования JavaScript

Язык программирования JavaScript также как и PHP разработан специально для ведения web разработки. Однако, в отличие от PHP, код JavaScript выполняется на стороне клиента, что позволяет генерировать динамические страницы html, не используя ресурсы сервера. JavaScript – это объектно-ориентированный язык, главным объектом которого является document. Данный объект имеет все необходимые методы (функции) и свойства для форматирования и тематического наполнения страниц html. Объект document также имеет ряд функций, позволяющих манипулировать непосредственно заголовками HTTP протокола (например, осуществлять управление Cookies).

Код JavaScript должен быть расположен на странице html между специальными тэгами **<script></script>.**

```
<HTML>
<HEAD>
<TITLE>Пример JavaScript</TITLE>
</HEAD>
<BODY>
<script>
document.write('Hello World');
</script>
</BODY>
</HTML>
```

Рисунок 7.1 – Пример использования JavaScript

В приведенном примере (рисунок 6.1) для вывода текста на странице используется метод *write*, ранее упомянутого объекта *document*. Доступ ко всем методам объекта document осуществляется через точку. Операторы в языке JavaScript отделяются друг от друга точкой с запятой. Язык программирования JavaScript, как уже отмечалось выше – это полноценный язык программирования. Это значит, что помимо манипуляций с объектом document, в языке есть все необходимые структуры: циклы, операторы ветвления и т.д. (их рассмотрение выходит за рамки данного методического пособия). В таблице 7.1 приведены некоторые полезные функции языка JavaScript.

Таблица 7.1 – Некоторые функции JavaScript

Метод	**Описание**
Math.abs(число)	Возвращает абсолютное значение числа
`alert("`*`сообщение`*`")`	Отображает диалоговое окно Alert с сообщением и кнопкой OK
`history.back()`	Позволяет вернуться на предыдущий URL в списке посещенных URL'ей.
`window.close()`	Закрывает указанное окно
`confirm("`*`сообщение`*`")`	Отображает диалоговое окно с указанным сообщением и кнопками OK и Cancel. Метод confirm используется для принятия пользователем решения, требующего выбора OK или Cancel. Аргумент *message* определяет сообщение, которое требует решения пользователя. Метод confirm возвращает true, если пользователь выбрал OK, и false, если пользователь выбрал

	Cancel.
`history.forward()`	Загружает следующий URL в списке посещенных URL'ей.
`document.write(`*`выражение1 [,выражение 2], ... [,выражениеN]`*`)`	Метод write отображает любое количество выражений в окне документа. Вы можете определить любое JavaScript выражение методом write, включая числовое, строковое или логическое.

8 Описание лабораторного web-сайта

Специально для проведения лабораторных работ был разработан web-сайт, позволяющий изучить основные классы web уязвимостей и способы защиты от них. Внешний вид сайта представлен на рисунке 8.1

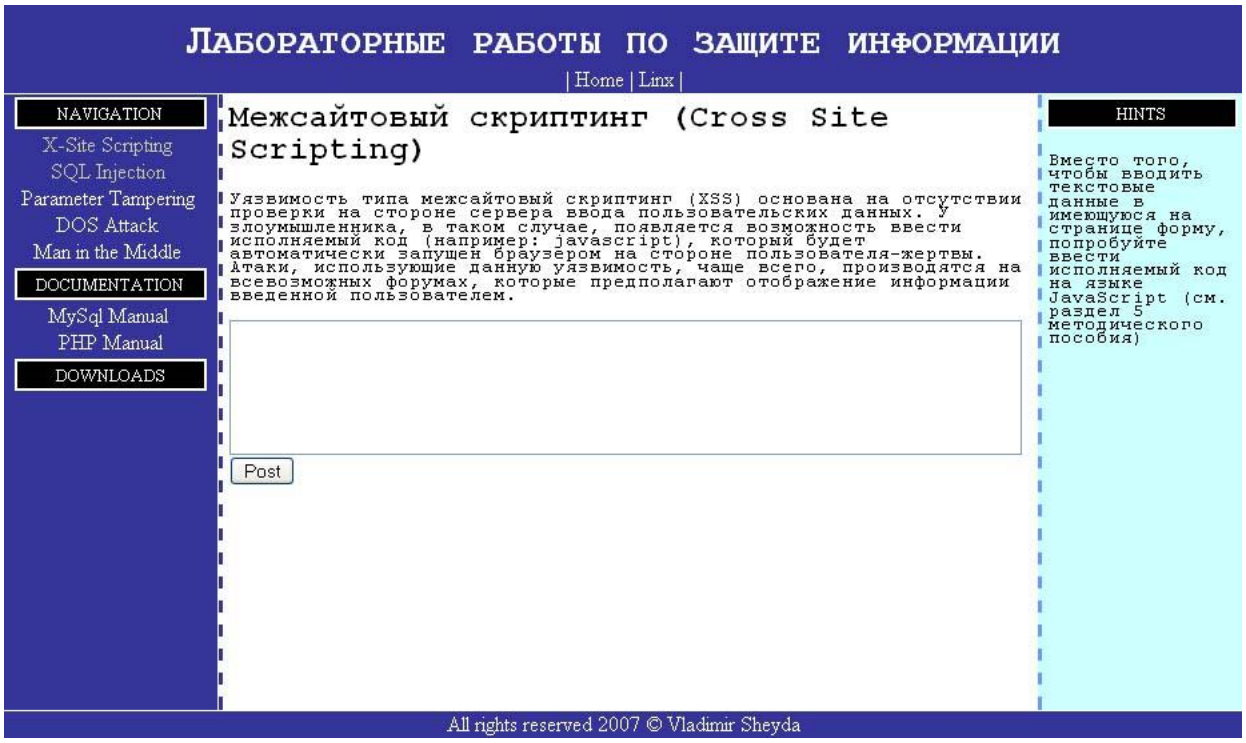

Рисунок 8.1 – Интерфейс лабораторного web-сайта

Web-сайт написан на языке PHP с использованием базы данных mySQL в качестве хранилища текстовой информации.

В таблице 8.1 перечислены файлы формирующие web-сайт.

Таблица 8.1 – файлы составляющие лабораторный web-сайт

Файл	**Назначение**
template.html	Главный шаблон web-интерфейса
jam.css	Каскадная таблица стилей
index.php	Скрипт страницы «Введение»
xss.php	Скрипт страницы «Межсайтовый скриптинг»
process.php	Скрипт обрабатывающий кнопку Post страницы «Межсайтовый скриптинг». Помещает текст, введенный в поля формы
param-tamp.php	Скрипт страницы «Подмена переменных»
tamper_me.php	Скрипт включенный в страницу «Подмена переменных» с помощью функции include. Передается в качестве значения переменной file методом GET
dos-attack.php	Скрипт страницы «Атака типа отказ в обслуживании»
mim.php	Скрипт страницы «Атака типа человек посредине»
functions.php	Набор функций для соединения с базой данных, получения информации из базы данных и записи произвольного файла в

	массив

Комментарии к скрипту xss.php

xss.php наиболее сложный скрипт данного web-сайта. Его основное назначение – вывод информации о рассматриваемой уязвимости и формы для ввода текстовых данных.

Обработкой данных введенных в форму и занесением их в соответствующую базу данных занимается скрипт process.php, запускаемый по нажатию на кнопку Post (рисунок 8.1). На рисунке 8.2 представлен листинг скрипта xss.php.

```
1  <?php
2  require_once('functions.php');

3  $tmpl=file2str('template.html');

4  $regexp=array('/%text%%/','/%hints%%/');

5  $form_placement="<FORM METHOD=POST ACTION=\"process.php\"> <TEXTAREA
   NAME=\"ui\" ROWS=\"5\" COLS=\"71\"> </TEXTAREA> <br> <INPUT
   TYPE=\"submit\" VALUE=Post> </FORM>";

6  $link=db_connect('main');
7  $content=db_get_text_data(2);

8  $maintext="<h2>".$content["title"]."</h2> <p>".$content["data"]."</p>";
9  $hinttext="<p style=\"padding: 5px;\" align=left
        >".$content["hints"]."</p>";

10 $query='select data from forum';
11 $result=mysql_query($query) or die('Mysql Error');
12 $query_output='';
13 while($row=mysql_fetch_array($result, MYSQL_ASSOC))
14 $query_output.="<hr noshade size=1px color=blue>".$row['data'];

15 mysql_free_result($result);
16 mysql_close($link);

17 $mainarea=$maintext.$form_placement.$query_output;
18 $hintarea=$hinttext;

19 $replacement=array();

20 $replacement[1]=$mainarea;
21 $replacement[2]=$hintarea;

22 $html=preg_replace($regexp,$replacement,$tmpl);
23 echo $html;
24 ?>
```

Рисунок 8.2 – Листинг скрипта xss.php

Строки 1 и 24 – открывающий и закрывающий тэги кода php
Строка 2 – подключение файла functions.php
Строка 3 – функция file2str, реализованная в functions.php принимает имя файла tmplate.html (главный шаблон сайта) и возвращает строку – содержимое этого файла. Строка сохраняется в переменной $tmpl.
Строка 4 – объявляется массив $regexp, в котором содержатся два элемента: `/%text%%/` и `/%hints%%/`.

Строка 5 – в переменную $form_placement помещается html код, отвечающий за вывод формы для ввода текстовых данных.
Строка 6 – функция db_connect принимает имя базы данных, с которой необходимо соединиться. Возвращенный функцией результат помещается в переменную $link, которая в дальнейшем будет использована как идентификатор данного соединения.
Строка 7 – функция db_get_text_data получает текстовую информацию, описывающую уязвимость «Межсайтовый скриптинг» из базы данных mySQL и помещает ее в переменную $content.
Строка 8 – в переменную $maintext помещается html код для вывода форматированных текстовых данных на главную часть страницы.
Строка 9 - в переменную $hinttext помещается html код для вывода форматированных текстовых данных в столбец HINTS страницы.
Строки 10 – 14 – происходит соединение с базой данных mySQL, из таблицы forum которой выбираются данные, введенные пользователем с помощью соответствующей формы на странице. Html-код, форматирующий эти данные, помещается в переменную $query_output.
Строки 15 и16 – функции mysql_close и myssql_free_result закрывают соединение с базой и очищают переменную $result.
Строка 17 – в переменную $mainarea помещаются html код (текстовая информация и форма) для вывода в основной части страницы.
Строка 18 – в переменную $hintarea помещаются html код для вывода на странице в столбце HINTS.
Строки 19 – 21 – формируют массив, состоящий из элементов приравненных к $mainarea и $hintarea.
Строка 22 – функция preg_replace заменяет строки $regexp в строке $tmpl на данные, хранящиеся в $replacement, и сохраняет результат в переменной $html.
Строка 23 – выводит переменную $html, тем самым формируя страницу которая отображается пользователю.

Комментарии к скрипту process.php

Скрипт process.php помещает текстовую информацию, введенную пользователем через форму н а странице xss.php в таблицу forum базы данных mySQL. process.php запускается кнопкой Post расположенной на странице xss.php. На рисунке 8.3 представлен листинг скрипта process.php.

```
1  <?php
2  require_once('functions.php');
3  $toins=$_POST['ui'];

4  $link=db_connect('main');
5  $query="insert into forum (data) values ('".$toins."')";

6  if(!$result=mysql_query($query)){header("Loacation: xss.php");}
7  //echo mysql_errno()."---".mysql_error();

8  mysql_close($link);
9  header("Location: xss.php");
10 ?>
```

Рисунок 8.3 – Листинг скрипта process.php

Строки 1 и 10 – открывающий и закрывающий тэги кода php.
Строка 2 – подключение файла functions.php.
Строка 3 – $_POST массив, содержащий переменные, переданные с помощью POST запроса. POST запрос в данном случае формируется при нажатии на кнопку Post на странице xss.php (см.

рисунок 8.2, строка 5). Данные, введенные в форму передаются в переменной ui, доступ к которой осуществляется по средствам массива $_POST и сохраняются в переменной $toins.
Строка 4 – функция db_connect принимает имя базы данных, с которой необходимо соединиться. Возвращенный функцией результат помещается в переменную $link, которая в дальнейшем будет использована как идентификатор данного соединения.
Строка 5 – формирование запроса SQL для помещения значения переменной $toins в таблицу forum столбец data базы данных mysql.
Строка 6 – функция mysql_query, выполняющая SQL запрос $query, возвращает «истину», если запрос был успешно выполнен. Структура if проверяет возвращенный функцией mysql_query результат и если результат «истина», то формирует заголовок HTTP Location, который перенаправляет браузер на страницу xss.php.

Комментарии к скрипту functions.php

Файл functions.php содержит реализации функций, которые позволяют подключиться к базе данных mySQL, получить текстовую информацию из базы а также записать содержимое файла в массив.

```
1  <?php
2  function file2str($file_name)
3  {
4  $lines=file($file_name); //Reads file into an array
5  $result=implode('',$lines); //Join array elements with a string
6  return $result;
7  }
8  function db_connect($db_name)
9  {
10 $link=mysql_connect('localhost','root','') or die('mysql error');
11 mysql_select_db('main') or die('could not select a db');
12 return $link;
13 }
14 function db_get_text_data($id)
15 {
16 $query="SELECT title, data, hints FROM text where id=$id";
17 $result=mysql_query($query) or die ('Bad query');
18 $row=mysql_fetch_array($result,MYSQL_ASSOC);
19 mysql_free_result($result);
20 return $row;
21 }
22 ?>
```

Рисунок 8.4 – Листинг скрипта functions.php

Строки 1 и 23 – открывающий и закрывающий тэги кода php.
Строка 2 – объявление функции file2str. Назначение функции – записать содержимое, какого либо файла в строку. Аргумент функции $file_name – имя файла, который необходимо представить в виде строки.
Строка 4 – стандартная функция PHP file записывает содержимое файла в массив $lines.
Строка 5 – стандартная функция PHP implode, конвертирует массив в строку. Результат сохраняет в переменной $result.
Строка 6 – возвращает переменную $result.
Строка 8 – объявление функции db_connect. Назначение функции – соединиться с базой данных mySQL. Аргумент функции $db_name – имя базы данных, к которой необходимо подсоединиться.
Строка 10 – стандартная функция PHP mysql_connect выполняет соединение с сервером mySQL. В качестве аргументов передаются: адрес сервера, пользователь, под которым осуществляется

соединение и пароль. Функция возвращает идентификатор соединения, который записывается в переменную $link.
Строка 11 - стандартная функция PHP mysql_select осуществляет выбор указанной в качестве аргумента функции базы данных.
Строка 12 – возвращает значение идентификатора соединения с сервером mySQL $link.
Строка 14 - объявление функции db_get_text_data, которая возвращает текстовую информацию, полученную из определенной таблицы базы данных mySQL. В качестве аргумента функция принимает $id – переменную, указывающую на строку в таблице из которой необходимо извлечь данные.
Строка 16 – переменная $query содержит запрос к базе данных, с помощью которого осуществляется выборка информации из столбцов title, data и hints в строке в которой id равен переменной $id переданной в качестве аргумента функции db_get_text_data.
Строка 17 - стандартная функция PHP mysql_query, выполняет SQL запрос к базе данных
Строка 18 - стандартная функция PHP mysql_fetch_array возвращает массив $row согласно запросу $query.
Строка 19 – очищает переменную $result.
Строка 20 – возвращает переменную $row.

9 Лабораторная работа №1: «Межсайтовый скриптинг»

9.1 Введение

Цель работы: использование уязвимости типа «Межсайтовый скриптинг» для проведения атаки на web-приложение и реализация защиты от данного типа атак.

Для успешного выполнения лабораторной работы необходимо детально изучить скрипты xss.php и process.php лабораторного web-сайта (см. раздел 8 данного методического пособия), что подразумевает понимание назначения скриптов в целом и знание всех используемых в данных скриптах функций.

9.2 Краткая теория

Уязвимость типа межсайтовый скриптинг (Cross-site scripting – XSS) предполагает использование злоумышленником web-приложения, для того чтобы передать исполняемый исходный код пользователю, использующему данное приложение.

Уязвимости такого типа широко распространены и используются в web-приложениях, в которых вводимые пользователем данные отображаются без их предварительной проверки.

Злоумышленник в таком случае способен послать вредоносный код ничего неподозревающему пользователю, браузер которого этот код исполнит. Вредоносный скрипт, запущенный браузером, в свою очередь, получит доступ ко всем cookies, идентификаторам сессий и другой конфиденциальной информации хранимой браузером.

Атаки, использующие уязвимость типа «межсайтовый скриптинг», делятся на две категории: **пассивные** и **активные**.

Пассивные атаки это атаки, которые хранятся непосредственно на «сервере-мишени», например, в базе данных, форуме, пользовательском логе и т.д. Жертва в таком случае получает вредоносный код при запросе сохраненной на сервере информации.

При проведении активных атак, web-сайт используется только как средство отображения вредоносного кода, в роли хранилища атаки выступают ссылки, передаваемые по средствам электронной почты либо публикуемые на сторонних web-сайтах. Пользователь, нажавший на подобную ссылку, перенаправляется на web-сайт отображающий вредоносный код, например, как результат поиска или сообщения об ошибке.

Последствия XSS атаки одинаковы, не зависимо от того пассивная она или активная. Единственное отличие заключается в том, каким способом вредоносный код достигает web-сервера. Распространено заблуждение, что сайт, на котором нет форм для ввода пользовательских данных, не содержит уязвимости типа XSS.

Межсайтовый скриптинг может стать причиной полной компрометации учетной записи жертвы. Наиболее опасные XSS атаки предполагают получение пользовательской cookie (см. раздел 2.4), хранящей идентификатор сессии, что позволяет злоумышленнику полностью завладеть учетной записью пользователя. Вместе с этим XSS атаки позволяют получить доступ к файлам жертвы и установке на стороне жертвы «Троянских коней», которые перенаправляют пользователей на другую страницу или сайт тем самым, модифицируя запрошенную информацию.

XSS уязвимость, позволяющая модифицировать информацию, например пресс релиз или новость, может повлиять на стоимость акций какой-либо компании или ввести пользователя в заблуждения, тем самым подорвав его доверие к данной компании.

Злоумышленники часто используют всевозможные способы кодирования вредоносных частей тэгов, например, используя Unicode, тем самым, делая запрос менее подозрительным.

Существуют сотни различных вариантов атак, использующих XSS, включая атаки, которые не используют символы “<”, “>”. Именно поэтому попытка отфильтровать подобные

скрипты не позволит решить проблему. Для определения легитимности введенных данных рекомендуется сравнивать их с набором разрешенных данных. Чаще всего XSS атаки реализованы с помощью встраиваемого JavaScript'а, хотя и любой встраиваемый исполняемый код, такой как ActiveX (OLE), VBscript, Shockwave, Flash и др., – потенциальный источник опасности.

Большинство web-серверов и серверов приложений генерируют web-страницы в случае возникновения различных ошибок HTTP, таких как 404 – «страница не найдена» или 500 «внутренняя ошибка сервера». Если такие страницы отображают какую-либо информацию из пользовательского запроса, например URL, который был запрошен, то они могут быть уязвимы для активных XSS атак.

Очень сложно найти и избавиться от данного типа уязвимости. Лучший способ сделать это – проанализировать весь код web-сайта и найти все места ввода пользовательских данных, которые затем могут быть отображены на странице, а, следовательно, интерпретированы браузером.

Необходимо помнить, что существует большое число HTML тэгов, которые могут быть использованы для передачи вредоносного JavaScript.

Для поиска уязвимостей также необходимо использовать сканеры такие как Nessus, Nikto и другие доступные инструменты.

Контрмеры

Лучший способ защитить web-приложение от XSS атаки – убедиться в том, что оно осуществляет проверку всех заголовков, cookies, строк запроса, скрытых полей и полей форм (например, всех параметров) на предмет соответствия жестким требованиям того, что разрешено. Не следует пытаться идентифицировать, а затем удалить либо отфильтровать исполняемый код, так как существует огромное количество разновидностей такого кода, и способов его кодирования.

Кодирование данных введенных пользователем также может защитить от XSS атак, делая вредоносный код неисполняемым. Приложения могут быть защищены от атак, базирующихся на JavaScript путем конвертирования метасимволов в соответствующие коды HTML (таблица 9.1).

Таблица 9.1 – Коды HTML

Символ	**HTML коды**
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&#39;
(	&#40;
)	&#41;
#	&#35;
%	&#37;
;	&#59;
+	&#43;
-	&#45;

Для выполнения такого конвертирования в PHP существует функция htmlentities (см. раздел 6 данного методического пособия)

9.3 Содержание работы

Данная лабораторная работа состоит из четырех этапов:

- изучение задействованных скриптов лабораторного web-сайта
- реализация атаки, использующей уязвимость «Межсайтовый скриптинг»
- реализация защиты от изучаемой уязвимости
- тестирование защиты

9.4 Порядок выполнения работы

1. Используя стандартный текстовый редактор, откройте файл xss.php, расположенный в корневой директории web-сервера.
2. Проанализируйте скрипт xss.php: необходимо изучить функциональное назначении каждой строки кода данного скрипта.
3. Используя стандартный текстовый редактор, откройте файл process.php, расположенный в корневой директории web-сервера.
4. Проанализируйте скрипт process.php: необходимо изучить функциональное назначении каждой строки кода данного скрипта.
5. С помощью браузера откройте страницу xss.php лабораторного web-сайта. Для этого перейдите по соответствующей ссылке из раздела Navigation главной страницы web-сайта.
6. Введите какой-либо текст в соответствующее поле формы и нажмите Post.
7. Найдите свою запись в базе данных. Для этого воспользуйтесь web-интерфейсом для управления базами данных phpMyAdmin (см. Приложение 2).
8. Используя браузер, в форму на странице xss.php введите код на языке JavaScript, выводящий в отдельном alert окне сообщение «Внимание! На сайте найдена XSS уязвимость» (см. раздел 7 данного методического пособия).
9. Найдите свою запись в базе данных. Для этого воспользуйтесь web-интерфейсом для управления базами данных phpMyAdmin (см. Приложение 2).
10. Перейдите к той части исходного кода xss.php и/или process.php, которая осуществляет вставку введенного в форме текста в базу.
11. Перепишите соответствующую строку кода, для того чтобы избежать запуска JavaScript на странице. Для этого необходимо использовать функцию языка PHP, конвертирующую метасимволы в коды HTML (см. раздел 6 данного методического пособия).
12. Удалите все записи из таблицы forum базы данных лабораторного web-сайта. Для этого воспользуйтесь web-интерфейсом для управления базами данных phpMyAdmin (см. Приложение 2).
13. Повторите пункты 6 – 9. Убедитесь в том, что JavaScript не выполняется.
14. С помощью браузера откройте исходный HTML код страницы. Найдите строку отображающую введенный вами JavaScript.

9.5 Контрольные вопросы

1. Основные компоненты Web технологии и их назначение.
2. Описать взаимодействие клиента (браузера) и сервера на уровнях протоколов IP, TCP и HTTP.
3. HTTP протокол: назначение, функции. Структура запроса и ответа. Основные методы и заголовки. Cookie: назначение и использование; проблемы безопасности связанные с cookies. (см. раздел 2 методического пособия).
4. Описать схему проведения и последствия атаки, использующей XSS уязвимость. Чем отличается пассивная XSS от активной? Определить тип XSS, использованной в лабораторной работе.
5. Как, используя PHP, соединиться с базой mySQL и выполнить какой-либо запрос?
6. Назначение функции PHP **htmlentities.**

10 Лабораторная работа №2: «SQL инъекция»

10.1 Введение

Цель работы: использование уязвимости типа «SQL-инъекция» для проведения атаки на web-приложение и реализация защиты от данного типа атак.

Для успешного выполнения лабораторной работы необходимо детально изучить скрипты sql-inj.php и functions.php лабораторного web-сайта (см. раздел 8 данного методического пособия), что подразумевает понимание назначения скриптов в целом и знание всех используемых в данных скриптах функций.

10.2 Краткая теория

Процесс взаимодействия web-сервера с базой данных

Вспомните архитектуру Web-приложений, описанную в главе 1. В этой главе рассматривается такой элемент этой структуры, как база данных. Рассмотрим, каким образом Web-сервер взаимодействует с базой данных. Поскольку Web-сервер работает только с протоколом HTTP, а сервер баз данных понимает только один язык — SQL, можно привести много примеров ситуаций, в которых Web-серверу нужно обращаться к базе данных, но здесь рассматривается самая очевидная — обработка регистрационных данных (имя и пароль) пользователя.

Когда пользователь входит в защищенную область Web-узла, Web-приложение осуществляет проверку соответствия его имени и пароля, которые чаще всего хранятся в базе данных, для чего приложение создает SQL-запрос, где в качестве параметров указаны пользовательское имя и пароль. Этот запрос вначале передается Web-серверу (например, с помощью сценария sql-inj.php) чтобы сервер осуществил проверку правильности комбинации "пользовательское имя — пароль". Для этого Web-сервер устанавливает соединение с базой данных. Соединение может быть установлено один раз и использоваться при дальнейших обращениях к серверу, а может устанавливаться и разрываться при каждом запросе. В любом случае для входа сервер использует собственные пользовательское имя и пароль.

После того, как Web-сервер подключился к базе данных, начинается обмен данными. В первую очередь с помощью сценария (sql-inj.php) в виде SQL-запроса базе данных передаются регистрационные данные пользователя (имя и пароль). База данных обрабатывает поступивший запрос, сообщает либо о совпадении данных запроса с одной из записей базы данных, либо о том, что таких совпадений не обнаружено. Затем Web-сервер с помощью сценария sql-inj.php обрабатывает ответ и, в зависимости от того, совпали или нет регистрационные данные с информацией, хранящейся в базе данных, предлагает пользователю продолжить работу или снова ввести пользовательское имя и пароль.

Язык структурированных запросов SQL является важной частью приложений. На сегодняшний день существует не так много средств, которые позволяют столь же эффективно работать с большими массивами данных, как это делают системы управления базами данных (СУБД). Вследствие этого в мире компьютерных технологий язык SQL получил широкое распространение. Вот почему как администратор, так и хакер должны понимать, как могут использоваться для взлома SQL-запросы.

Техника выполнения атаки использующей уязвимость типа SQL инъекция

Уязвимость, называемая SQL инъекцией (SQL injection), может реализовываться как в виде простого внедрения символов, вызывающих ошибки, так и в виде выполнения произвольных запросов из командной строки. Продукты практически всех поставщиков баз данных могут подвергаться взломам данного типа. Этот изъян проявляется в SQL-запросах и

соответствующих программных интерфейсах, предназначенных для поддержки таких запросов: ASP, PHP, Perl, а также любой другой язык программирования для Web.

При введении условия SQL-запросов чаше всего используется символ одинарной кавычки (‘). Поэтому для поиска возможности взлома можно, прежде всего, попытаться использовать этот символ. В структуре SQL-запроса одинарная кавычка обычно используется для отделения переменных, передаваемых в запросе от самого запроса.

```
strSQL =   "select userid from users where password =   ' "  + password + " ' ";
```

В таблице 10.1 приведен список других часто употребляемых символов, предназначенных для форматирования SQL-запросов. С помощью таких символов можно осуществлять проверку SQL-сервера на наличие изъянов. Проверка Web-узла с использованием этих символов позволяет найти изъяны SQL-сервера без выполнения хранимых процедур или сложных SQL-запросов.

Таблица 10.1 – часто употребляемые символы для форматирования SQL-запросов

Управляющие символы SQL	**Описание**
‘	Завершает выражение
--	Однострочный комментарий. Оставшаяся часть строки игнорируется
+	Пробел. Требуется для корректного формата выражений
,@переменная	Добавляет переменную. Помогает идентифицировать хранимые процедуры
?Param1=foo&Param1=bar	Присваивает переменной Param1 значения foo и bar. Помогает идентифицировать хранимые процедуры
@@переменная	Обращение к внутренней переменной сервера
PRINT	Возвращает ошибку ODBC, но не может служить для вывода данных
SET	Объявление переменных. Используется для многострочных SQL-выражений
%	Символ шаблона, который заменяет любую строку, состоящую из нулевых или других символов

Операторы SQL

Язык SQL состоит из списка предопределенных ключевых слов и специальных символов. Для того чтобы извлечь какие-либо данные из таблицы, необходимо воспользоваться оператором SELECT. При работе с Web-приложениями, как правило, используется именно оператор SELECT с ключевыми словами from и WHERE. Внедряя инструкции SQL в такие запросы, хакер может заставить запрос выдать другую информацию или, например, сделать так, чтобы при проверке на какое-либо условие приложению возвращался положительный ответ.

Формат SQL-выражений строго не определен, и одно и то же выражение может быть записано несколькими способами, что очень усложняет восприятие инструкций языка. Эти особенности языка могут помочь хакеру получить доступ к базе данных. Естественно, что для получения более серьезных результатов необходимо использовать более сложные структуры запросов, но, тем не менее, все эти взломы основаны на этих же основополагающих принципах.

Вероятность обнаружения изъяна базы данных повышает использование разнообразных методов проверки — начиная от внедрения одинарной кавычки и двойного тире в SQL-запрос и

заканчивая множественными кавычками и круглыми скобками. Для получения доступа к базе данных, вероятнее всего, придется перепробовать все эти методы.

OR 1=1

Очевидно, что это выражение создает логическое выражение, результатом вычисления которого всегда будет "Истина". Данное выражение используется в запросах аутентификации, которые проверяют имя пользователя и пароль.

```
sqlAuth = "SELECT userid FROM logins WHERE name='" & Username & "' AND password='" & Password & "'"
```

Если пользователь входит в систему под именем Wayne с паролем Pirate, то запрос будет выглядеть следующим образом.

```
SELECT userid FROM logins WHERE name='Wayne' AND password='Pirate'
```

Таким образом, пользователь Wayne не сможет войти в систему до тех пор, пока значение Pirate не попадет в базу данных. Однако, применяя выражение OR 1=1, можно избавиться от проверки пароля.

```
SELECT userid FROM logins WHERE name='Wayne' AND password='Pirate' OR 1=1
```

UNION

Оператор UNION используется совместно с оператором select и служит для извлечения всех столбцов из таблицы. Синтаксис использования команды следующий.

```
UNION ALL SELECT field FROM table WHERE condition
```

Кроме того, этот оператор может использоваться для извлечения имен полей и таблиц из названий переменных в приложении, файлах .inc или сообщений об ошибках SQL. Условие 1=1 или "=" (ничего равняется ничему) всегда истинно.

INSERT

Как несложно догадаться, инструкция insert предназначена для добавления каких-либо значений в таблицу базы данных. На первый взгляд данная команда выглядит не слишком полезной с точки зрения взлома; в конце концов, мы же хотим узнать содержимое базы данных, а не наполнять ее информацией. Однако не стоит поддаваться первым впечатлениям — эта команда может быть прекрасно использована для прохождения аутентификации незарегистрированного пользователя. Вот как, например, с помощью внедрения SQL, можно добавить в таблицу пользователей Users нового пользователя с именем neo и паролем trinity.

```
INSERT INTO Users VALUES('neo', 'trinity')
```

Аутентификационные данные для получения доступа к базе данных. Чтобы присоединиться к базе данных, Web-сервер должен знать пользовательское имя и пароль. Данное соединение производится сервером в автоматическом режиме. Следовательно, приложения хранят регистрационные данные для аутентификации в теле какой-либо страницы. Но, к сожалению, большинство приложений помещают эти данные в какой-нибудь файл, расположенный в корневом каталоге Web-сервера, — место, к которому обычно имеют доступ Web-браузеры пользователей.

Некоторые разработчики приложений надеются, что сервер защитит их важные файлы, как это делает сервер IIS, запрещающий запросы к файлу global.asa. Однако, если приложение подвержено опасности раскрытия исходного кода (что в случае с Web-приложениями встречается довольно часто), то имя пользователя и пароль Web-сервера могут быть похищены.

Кроме того, разработчики иногда помещают эти регистрационные данные в файлы, которые, как они считают, пользователь не будет просматривать или при просмотре просто не найдет этих данных. Файлы могут иметь такие имена, как xmlserver.js, database.inc или server.js.

Однако эти строки легко обнаружить, если в них используется ключевое слово password.

```
strConn = "Provider=SQLOLEDB;Data Source=dotcomdb;
Initial Catalog=Demo;
User Id=sa; Passwords="
```

Файл СУБД Oracle global.asa также может содержать регистрационные данные.

Общие контрмеры

Практически каждая база данных имеет средства для безопасной установки и безопасного хранения данных. Инициализировав данные службы, можно защитить вашу базу данных от взломов с использованием внедрения SQL-инструкций на уровне приложения.

Правильная обработка ошибок

Никогда не позволяйте приложению отсылать пользователю сообщения об ошибках механизма ODBC или любых других ошибках. Используйте генерацию страниц, описывающих причину ошибки, но ни в коем случае не предоставляйте пользователю системную информацию, имена переменных или другие данные. В языке Java, например, для обработки ошибок очень удобным будет использование методов try, catch, finally.

Список параметров

Данные, относящиеся к пользователю, необходимо помещать в специальные переменные. Объединение строк представляет серьезную угрозу для безопасности SQL-выражений, поскольку обеспечивает пользователю простой способ манипулирования выражением с помощью добавления кавычек.

Проверка правильности ввода должна осуществляться на Web сервере, а базе данных должны передаваться только данные со строго заданными параметрами. Поля, в которые пользователь осуществляет ввод только численных данных, должны иметь тип только INT и ни в коем случае не VARCHAR.

Хранимые процедуры

Хотя и нельзя утверждать со стопроцентной уверенностью, но хранимые процедуры, которые были определены пользователем, менее подвержены опасности взлома с использованием внедрения SQL. Они требуют строго определенного количества параметров, в строго определенной последовательности и в строго определенном формате. Эти жесткие требования значительно уменьшают риск взлома данной процедуры. Кроме того, использование хранимых процедур повышает не только уровень безопасности приложения, но также улучшает его характеристики при использовании!

Выполнение с минимальными привилегиями

Все приложения, работающие с базами данных, должны выполняться в условиях минимальных привилегий. Также учетная запись пользователя, которую использует Web-сервер, должна иметь ограниченные возможности. Это не означает, что приложение не может производить запись и чтение из базы данных, однако данное приложение не должно иметь прав выполнения записи в главную базу данных или прав выполнения резервного копирования базы данных.

О Защита схемы

Имена таблиц, имена столбцов и SQL-структуры не должны размещаться в HTML-тексте. Часто встречаются страницы, на которых разработчики помещают объявление всей таблицы между HTML-дескрипторами комментария. Конечно же, эти действия облегчают понимание страниц, однако комментарии все же лучше помещать между ASP-дескрипторами комментария, которые может просматривать только разработчик, чтобы пользователь не имел к ним доступа.

10.3 Содержание работы

Данная лабораторная работа состоит из четырех этапов:

- изучение задействованных скриптов лабораторного web-сайта
- реализация атаки, использующей уязвимость «SQL инъекция»
- реализация защиты от изучаемой уязвимости
- тестирование защиты

10.4 Порядок выполнения работы

1. Используя стандартный текстовый редактор, откройте файл sql-inj.php, расположенный в корневой директории web-сервера.
2. Проанализируйте скрипт sql-inj.php: необходимо изучить функциональное назначении каждой строки кода данного скрипта (см. разделы 5 и 6 методического пособия).
3. С помощью браузера откройте страницу sql-inj.php лабораторного web-сайта. Для этого перейдите по соответствующей ссылке из раздела Navigation главной страницы web-сайта.
4. В поле Login введите *Koba*, а в поле Password – *lenin*. Запишите выведенную информацию. Какой SQL запрос выполняется при выводе информации из базы данных? Найдите в sql-inj.php строку, отвечающую за выполнение этого запроса.
5. Проведите атаку, использующую уязвимость типа «SQL инъекция». Для этого необходимо ввести в поля Login и Password значения, модифицирующие запрос SQL таким образом, чтобы логическая часть запроса была всегда истиной, т.е. необходимо использовать выражение OR 1=1, а также учесть кавычки (‘), которые должны быть закрыты, чтобы запрос был верен с точки зрения синтаксиса SQL.
6. Используя phpMyAdmin (Приложение 2), убедитесь в том, что с помощью проведенной атаки были выведены все данные, хранящиеся в соответствующей таблице базы данных main.
7. Перепишите код sql-inj.php таким образом, чтобы введенные пользователем символы кавычек заменялись соответствующими кодами html (см. разделы 6 и 9 данного методического пособия).
8. Повторите пункт 5. Убедитесь в том, что данные, хранящиеся в базе, не выводятся.

10.5 Контрольные вопросы

1. Составьте запрос на языке SQL, который бы вставлял следующую запись в таблицу users: в поле nickname значение “boba”, в поле password – “pwd”, а в поле email – boba@hostname.com
2. Составьте запрос на языке SQL, который бы выводил информацию, помещенную в поля nickname и email только той записи, в которой в поле passwd хранится значение “pwd”.
3. Модифицируйте предыдущий запрос таким образом, чтобы часть запроса после ключевого слова where была всегда истинной.
4. Объясните суть атаки, использующей уязвимость типа SQL инъекция, рассмотренной в лабораторной работе.
5. Какая функция была использована для защиты web-сайта от атак, использующих уязвимость типа SQL-инъекция.

11 Лабораторная работа №3: «PHP-include»

11.1 Введение

Цель работы: использование уязвимости типа «PHP-include» для проведения атаки на web-приложение и реализация защиты от данного типа атак.

Для успешного выполнения лабораторной работы необходимо детально изучить скрипты param-tamp.php, tamper_me.php и functions.php лабораторного web-сайта (см. раздел 8 данного методического пособия), что подразумевает понимание назначения скриптов в целом и знание всех используемых в данных скриптах функций.

11.2 Краткая теория

Уязвимость php-include одна из самых известных, но между тем одна из самых распространённых уязвимостей встречающихся сегодня в php сценариях (скриптах). Она возникает, когда по невнимательности либо по незнанию, программист позволяет использовать данные, переданные сценарию в качестве параметра функции include или require (см. раздел 6.2 данного методического пособия), без дополнительной проверки. Для того чтобы лучше разобраться в принципе действия данной уязвимости, необходимо иметь некоторое представление о вышеназванной функции.

Функция include (или include_once) используется для подключения к запущенному php сценарию дополнительных программных модулей. Причём, в отличие от похожей по свойствам функции require, функция include исполняет эти модули непосредственно в своём процессе. А, следовательно, прикрепляемые таким образом модули будут исполняться не как отдельные сценарии, а как части подключившего их к себе сценария. Функция include будет исполнять только ту часть файла, которая заключена между специальными тэгами, обозначающими начало (**<?php**) и конец (**?>**) сценария php. Всё остальное php просто выдаёт в виде текста. Т.е. если подключить текстовый файл (например, /etc/passwd), не содержащий указанных тэгов, всё содержимое этого файла будет выдано интерпретатором.

Пример вызова функции:

```
include($file);
```

Функция include имеет всего 1 параметр ($file), который указывает путь и имя файла подключаемого модуля. Стоит отметить также, что в Юникс-системах (в зависимости от настроек php) в качестве параметра можно передавать не только путь и имя файла, но и url (Интернет адрес) файла.

Предположим, на некотором web-сервере установлен следующий php сценарий (его url http://www.foo.com/index.php):

```
<?php
include($_GET["file"]);
?>
```

А также множество различных подключаемых сценариев-модулей для него:

home.php

feedback.php

...

Разработчиком предполагалось, что все посетители сайта будут переходить от одной страницы сайта к другой, следуя расположенным на сайте ссылкам. Сценарий в свою очередь в

зависимости от переданного параметра file, будет присоединять тот или иной модуль, таким образом, генерируя различные html страницы (чаще всего include используют именно таким образом).

Примеры запросов (жирным шрифтом выделены параметр и его значение, которое будет передано в качестве аргумента функции include):

http://www.superpupersite.com/index.php?**file=home.php**

http://www.superpupersite.com/index.php?**file=feedback.php**

Внимательно изучив данные ссылки, злоумышленник мог бы предположить, что значение параметра file это имя и путь к соответствующему файлу и, что сценарий использует функцию include для подключения этого файла.

Проверить это можно, попытавшись получить доступ, например, к файлу passwd, хранящемуся в каталоге /etc.

http://www.superpupersite.com/index.php?**file=/etc/passwd**

Помимо доступа к файлам находящимся на диске сервера, функция include предоставляет возможность запускать скрипты, находящиеся на удаленных рабочих станциях. Например, передав в качестве значения переменной адрес на следующий сценарий: **<?php passthru("ls -al"); ?>** - позволит злоумышленнику получить список всех файлов текущего каталога сервера. Допустим, что вышеупомянутый сценарий был сохранен в файле cmd.txt на web-сайте злоумышленника zlo.com, то, используя http://zlo.com/cmd.txt в качестве значения переменной **file**, можно запустить данный сценарий на стороне web-сервера жертвы:

http://www.superpupersite.com/index.php?**file=http://zlo.com/cmd.txt**

Контрмеры

Для предотвращения запуска удаленных скриптов необходимо создать список всех файлов (модулей), которые разрешено включать функцией include. И в зависимости, от того находится ли тот или иной модуль в списке, выполнять его, либо выдавать соответствующую ошибку. Пример использования конструкции switch() показан на рисунке 11.1.

```
switch ($filename)  // $filename - имя переменной, передаваемой в
                    // параметре к скрипту
{
case 'news': include("news.php");
break;

case 'articles': include("guestbook.php");
break;

... // и т.д.
default:
include("index.php"); // если в переменной $filename не будет передано
                      // значение, которое учтено выше, то открывается
                      // главная страница
break;
}
```

Рисунок 11.1 – Синтаксис switch()

11.3 Содержание работы

Данная лабораторная работа состоит из четырех этапов:

- изучение задействованных скриптов лабораторного web-сайта

- реализация атаки, использующей уязвимость «SQL инъекция»
- реализация защиты от изучаемой уязвимости
- тестирование защиты

11.4 Порядок выполнения работы

1. Используя стандартный текстовый редактор, откройте файл param-tamp.php, расположенный в корневой директории web-сервера.
2. Проанализируйте скрипт param-tamp.php: необходимо изучить функциональное назначении каждой строки кода данного скрипта.
3. Используя стандартный текстовый редактор, откройте файл tamper_me.php, расположенный в корневой директории web-сервера.
4. Проанализируйте скрипт tamper_me.php: необходимо изучить функциональное назначении каждой строки кода данного скрипта.
5. С помощью браузера откройте страницу param-tamp.php лабораторного web-сайта. Для этого перейдите по соответствующей ссылке из раздела Navigation главной страницы web-сайта.
6. Измените URI таким образом, чтобы в функцию include сценария param-tamp.php передался файл pass.txt, находящийся в текущей директории web-сервера.
7. Модифицируйте файл param-tamp.php таким образом, чтобы при подмене значения переменной в строке URI в любом случае функцией include выполнялся файл tamper_me.php. Используйте для этого структуру языка PHP switch (см. разделы 6 и 11.2 данного методического руководства)
8. Заново выполните пункт 6. Убедитесь в том, что независимо от того какое значение присваивается параметру file в URI, функция include получает и обрабатывает сценарий tamper_me.php

11.5 Контрольные вопросы

1. Перечислить все функции языка php, позволяющие включать в основной скрипт дополнительные сценарии.
2. Как в php осуществляется доступ к переменным передаваемым в строке запроса GET?
3. Какая конструкция языка php позволяет заменить оператор *if,* в случае большого количества различных условий? Синтаксис данной конструкции.
4. Объяснить суть уязвимости PHP-include.
5. Каким образом оператор *switch* позволяет защитить web-сайт от атаки, использующей уязвимость типа PHP-include?

12 Лабораторная работа №4: «Прослушивание трафика (Traffic sniffing)»

12.1 Введение

Цель работы: реализация перехвата трафика, передаваемого по сегменту сети. Изучение протокола TCP.

Для успешного выполнения лабораторной работы необходимо изучить структуру заголовка и механизмы установления и разрыва соединения протокола TCP, (см. раздел 3 методического пособия)

12.2 Краткая теория

В локальной вычислительной сети сниферы являются высокоэффективным средством для получения паролей и другой критической информации. Большинство обычно используемых протоколов либо передают данные явным образом (так называемый plaintext, который может быть легко перехвачен), либо не используют достаточно стойкую криптографию для предотвращения перехвата и расшифровки. Примерами протоколов, передающих данные явным образом, являются SMTP, POP3, SNMP, FTP, TELNET и HTTP.

Перехват трафика может осуществляться:

- обычным «прослушиванием» сетевого интерфейса (метод эффективен при использовании в сегменте концентраторов (хабов) вместо коммутаторов (свитчей) в противном случае метод малоэффективен, поскольку на снифер попадают лишь отдельные фреймы);
- подключением снифера в разрыв канала;
- ответвлением (программным или аппаратным) трафика и направлением его копии на снифер;
- через анализ побочных электромагнитных излучений и восстановление таким образом прослушиваемого трафика;
- через атаку на канальном (MAC-spoofing) или сетевом уровне (IP-spoofing), приводящую к перенаправлению трафика жертвы или всего трафика сегмента на снифер с последующим возвращением трафика в надлежащий адрес.

Сниферы применяются как для получения служебной информации для анализа работы сети (например, системными администраторами), так и для сбора информации, такой как пароли с целью проведения атаки на узлы сети. Анализ прошедшего через снифер трафика позволяет:

- Обнаружить паразитный, вирусный и закольцованный трафик, наличие которого увеличивает загрузку сетевого оборудования и каналов связи (сниферы здесь малоэффективны; как правило, для этих целей используют сбор разнообразной статистики серверами и активным сетевым оборудованием и ее последующий анализ).
- Выявить в сети вредоносное и несанкционированное ПО, например, сетевые сканеры, флудеры, троянские программы, клиенты пиринговых сетей и другие (это обычно делают при помощи специализированных сниферов — мониторов сетевой активности).
- Перехватить любой незашифрованный (а порой и зашифрованный) пользовательский трафик с целью получения паролей и другой информации.
- Локализовать неисправность сети или ошибку конфигурации сетевых агентов (для этой цели сниферы часто применяются системными администраторами).

Контрмеры

Основным решением, препятствующим перехвату трафика, является шифрование передаваемых данных. Так, в сети Интернет для доступа к сайтам с конфиденциальной информацией вместо обычного протокола HTTP используют протокол HTTPS, который использует SSL (Security Sockets Layer – протокол защищенных сокетов) для криптографической защиты передаваемой информации.

12.3 Содержание работы

Данная лабораторная работа состоит из двух этапов:

- перехват трафика с использованием снифера WireShark
- анализ перехваченных данных

12.4 Порядок выполнения работы

1. Запустите программу WireShark. Настройте её на перехват трафика на локальном сетевом адаптере. В качестве значения фильтра укажите port 80.
2. С помощью браузера зайдите на сайт sibmail.com и введите следующие данные:
   Логин:
   Пароль:
3. Нажмите кнопку Login и закройте браузер.
4. Остановите снифер и сохраните результат.
5. Проанализируйте собранную информацию. Укажите:
   a) порядковые номера фреймов с установленными в них флагами, которые участвовали в установлении соединения TCP
   b) порядковые номера фреймов, подтверждающих доставку
   c) порядковые номера фреймов с установленными в них флагами, которые участвовали в разрыве соединения TCP
   d) номер фрейма, в котором были переданы логин и пароль
   e) все протоколы, инкапсулированные в кадр Ethernet при посылке HTTP GET запроса и соответствующие им уровни согласно таблице OSI (см. Приложение 1)
6. Используя функцию Follow TCP Stream программы WireShark, найдите HTTP запрос, с помощью которого отсылается логин и пароль. Укажите метод запроса и все переданные в этом запросе параметры с их значениями.

12.5 Контрольные вопросы

1. Опишите процесс установления соединения TCP.
2. Опишите процесс разрыва соединения TCP.
3. Для чего в лабораторной работе необходимо установить значение фильтра программы-снифера port 80?
4. Как осуществляется гарантия доставки в протоколе TCP?
5. Перечислите все флаги протокола TCP и укажите их назначение.
6. Что такое порт, и каково его значение при передаче данных между двумя рабочими станциями?
7. Что такое окно протокола TCP и каково его назначение?

Литература

1. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – СПб., 2008. – 957 с.: ил.
2. Кришнамурти Б., Рексфорд Дж. Web-протоколы. Теория и практика. – М.: ЗАО «Издательство БИНОМ», 2002 г. – 592 с.: ил.
3. Красюк Д. Основы языка разметки гипертекста – http://about-html.narod.ru/index.htm
4. Паркер Т., Сиян К. TCP/IP. Для профессионалов. 3-е изд. — СПб.: Питер, 2004.— 859 с: ил.
5. Семенов Ю. А. Протоколы и ресурсы Internet М: Радиосвязь, 1996.
6. Скембрей, Джоел, Шема, Майк. Секреты хакеров. Безопасность Web-приложений — готовые решения. : Пер. с англ. — М.: Издательский дом "Вильяме", 2003. — 384 с.: ил. — Парал. Тит. англ.
7. Стивенс У. UNIX: Разработка сетевых приложений. – СПб.: Питер, 2004 – 1086 с.: ил. – (Серия «Мастер-класс»)
8. Чернов М. PHP include уязвимость: от теории к практике // Internet Zone http://www.izcity.com/data/security/article1001.htm, 2005
9. Open Web Application Security Project (OWASP) – http://owasp.org
10. MySQL AB::Developer Zone – http://dev.mysql.com/
11. PHP: Hypertext Preprocessor – http://www.php.net/
12. RFC документы – http://www.ietf.org/rfc.html
13. Security Focus – http://www.securityfocus.com/
14. Security Lab – http://www.securitylab.ru/
15. Wikipedia Free Encyclopedia – http://wikipedia.org

Приложение А

Модель OSI

Модель OSI (Open Systems Interconnection Reference Model — модель взаимодействия открытых систем) — абстрактная модель для сетевых коммуникаций и разработки сетевых протоколов представляет уровневый подход к сети. Каждый уровень обслуживает свою часть процесса взаимодействия. Благодаря такой структуре совместная работа сетевого оборудования и программного обеспечения становится гораздо проще и понятнее.

В 1978 году Международный комитет по стандартизации (ISO) разработал стандарт архитектуры ISO 7498, для объединения различных сетей. В разработке участвовало 7 комитетов, каждому из них был отведён свой уровень. В 1980 году IEEE опубликовал спецификацию 802, детально описавшую механизмы взаимодействия физических устройств на канальном и физическом уровнях модели OSI. В 1984 году спецификация модели OSI была пересмотрена и принята как международный стандарт для сетевых коммуникаций.

Уровни модели OSI

Модель состоит из 7-ми уровней, расположенных вертикально друг над другом. Каждый уровень может взаимодействовать только со своими соседями и выполнять отведённые только ему функции.

Таблица 11 – распределение сетевых протоколов по уровням модели OSI

Уровень OSI	**Протоколы**
Прикладной (7-ой уровень)	HTTP, gopher, Telnet, DNS, SMTP, SNMP, CMIP, FTP, TFTP, SSH, IRC, AIM, NFS, NNTP, NTP, SNTP, XMPP, FTAM, APPC, X.400, X.500, AFP, LDAP, SIP, ITMS, Modbus TCP, BACnet IP, IMAP, POP3, SMB, MFTP, BitTorrent, eD2k, PROFIBUS, NCP
Представления[2] (6-ой уровень)	MIME, XDR, XML
Сеансовый[3] (5-ый уровень)	ASP, ADSP, DLC, Named Pipes, NBT, NetBIOS, NWLink, Printer Access Protocol, Zone Information Protocol, SSL, TLS, SOCKS
Транспортный (4-ый уровень)	TCP, UDP
Сетевой (3-ий уровень)	IP, IPv6, ICMP, IGMP, IPX, NWLink, NetBEUI, DDP, IPSec, ARP, RARP, DHCP, BootP, SKIP, RIP
Канальный (2-ой уровень)	STP, ARCnet, ATM, DTM, SLIP, SMDS, Ethernet, FDDI, Frame Relay, LocalTalk, Token ring, StarLan, L2F, L2TP, PPTP, PPP, PPPoE, PROFIBUS
Физический (1-ый уровень)	RS-232, RS-422, RS-423, RS-449, RS-485, ITU-T, xDSL, ISDN, T-carrier (T1, E1), Ethernet (10BASE-T, 10BASE2, 10BASE5), Fast Ethernet (100BASE-T, 100BASE-TX, 100BASE-T4, 100BASE-FX)), Gigabit Ethernet (1000BASE-T, 1000BASE-TX, 1000BASE-SX)

Прикладной уровень (Application layer)

[2] Уровень объединен с прикладным
[3] Уровень объединен с прикладным

Верхний (7-й) уровень модели, обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя доступ к сетевым службам, таким как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты; а также отвечает за передачу служебной информации, предоставляет приложениям информацию об ошибках и формирует запросы к уровню представления.

Уровень представления (Presentation Layer)

Этот уровень отвечает за преобразование протоколов и кодирование/декодирование данных. Запросы приложений, полученные с уровня приложений, он преобразует в формат для передачи по сети, а полученные из сети данные преобразует в формат, понятный приложениям. На этом уровне может осуществляться сжатие/распаковка или кодирование/декодирование данных, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально.

Сеансовый уровень (Session Layer)

Отвечает за поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений. Синхронизация передачи обеспечивается помещением в поток данных контрольных точек, начиная с которых возобновляется процесс при нарушении взаимодействия

Транспортный уровень (Transport Layer)

Четвертый уровень модели, предназначен для доставки данных без ошибок, потерь и дублирования в той последовательности, как они были переданы. При этом неважно, какие данные передаются, откуда и куда, то есть он предоставляет сам механизм передачи. Блоки данных он разделяет на фрагменты, размер которых зависит от протокола, короткие объединяет в один, а длинные разбивает. Протоколы этого уровня предназначены для взаимодействия типа точка-точка.

Сетевой уровень (Network Layer)

Третий уровень сетевой модели OSI, предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию пакетов, отслеживание неполадок и заторов в сети. На этом уровне работает такое сетевое устройство, как маршрутизатор.

Канальный уровень (Data Link layer)

Этот уровень предназначен для обеспечения взаимодействия сетей на физическом уровне и контроля за ошибками, которые могут возникнуть. Полученные с физического уровня данные он упаковывает в кадры данных, проверяет на целостность, если нужно исправляет ошибки и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием. Спецификация IEEE 802 разделяет этот уровень на 2 подуровня — MAC (Media Access Control) регулирует доступ к разделяемой физической среде, LLC (Logical Link Control) обеспечивает обслуживание сетевого уровня. На этом уровне работают коммутаторы, мосты.

В программировании этот уровень представляет драйвер сетевой платы, в операционных системах имеется программный интерфейс взаимодействия канального и сетевого уровня между собой, это не новый уровень, а просто реализация модели для конкретной ОС.

Физический уровень (Physical Layer)

Самый нижний уровень модели, предназначен непосредственно для передачи потока данных. Осуществляет передачу электрических или оптических сигналов в кабель и соответственно их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов. Другими словами, осуществляет интерфейс между сетевым носителем и сетевым устройством. На этом уровне работают концентраторы, повторители (ретрансляторы) сигнала и сетевые адаптеры.

Приложение Б

Краткое описание phpMyAdmin

PhpMyAdmin используется для редактирования и просмотра информации хранимой в базах данных какого-либо сервера.

Для запуска phpMyAdmin необходимо в браузере ввести следующий URL: http://[IP-адрес сервера]/phpmyadmin. Для входа необходимо ввести в поле Login открывшейся страницы имя пользователя ‘root’, поле пароль оставить пустым и нажать на кнопку Пошел.

Выбор базы данных осуществляется в выпадающем меню в левой части экрана (рисунок 12)

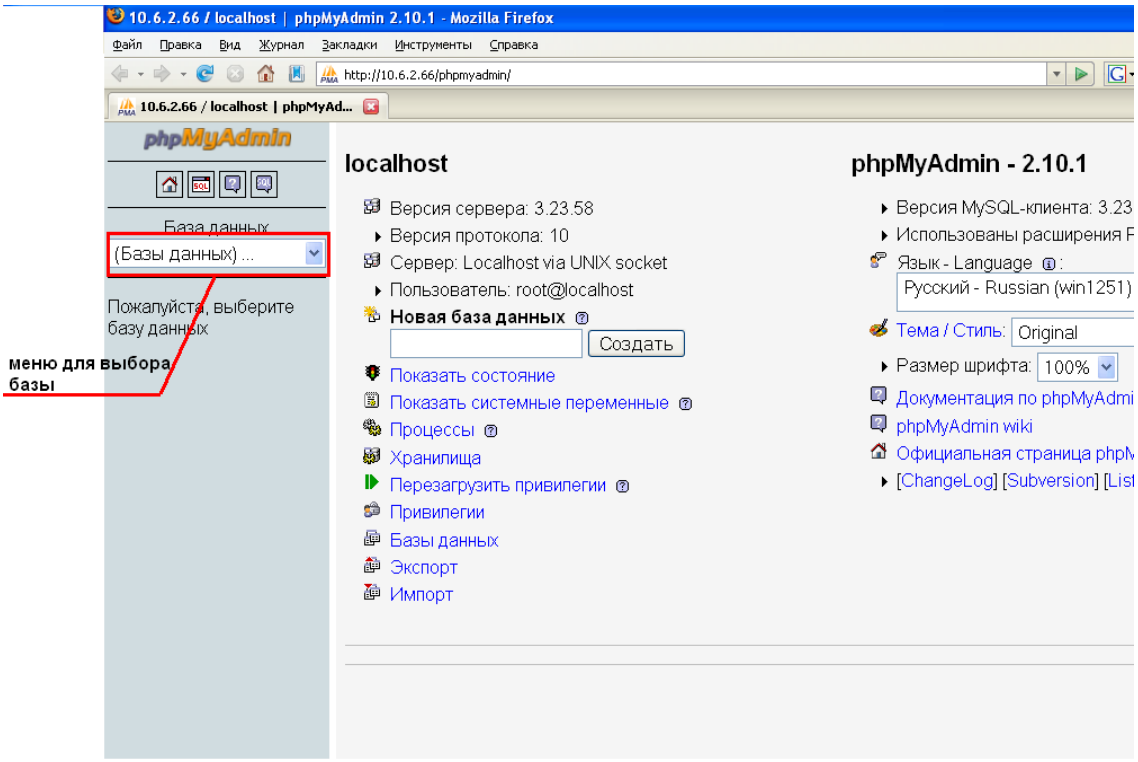

Рисунок 12 – выбор базы данных в phpMyAdmin

После того как база выбрана в главной части экрана появится список таблиц данной базы данных (рисунок 13).

Для выполнения какого-либо действия над таблицей, например просмотр хранимых данных, удаление данных из таблицы, удаление таблицы и др., необходимо нажать на соответствующей иконке в разделе действия (рисунок 13)

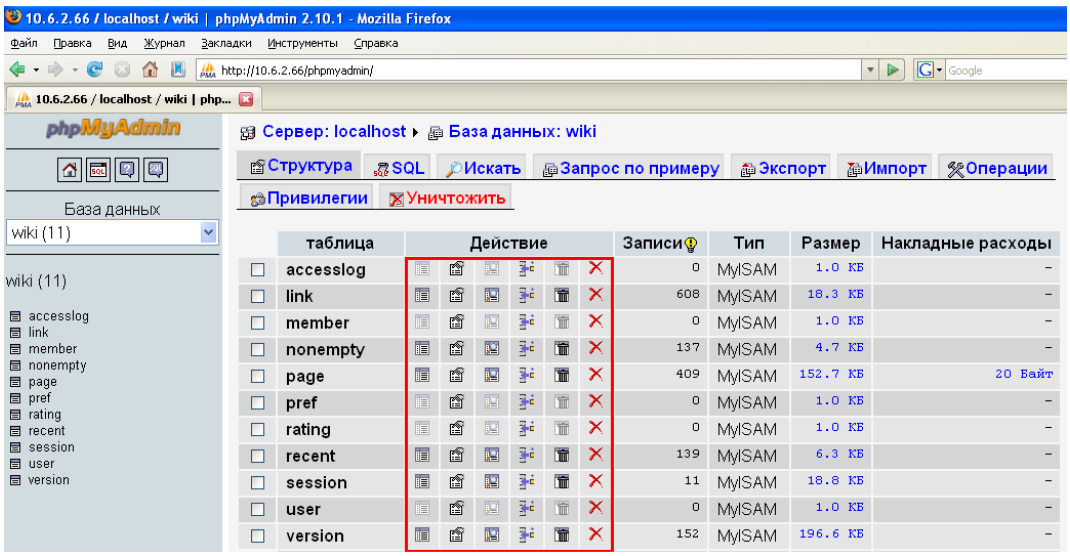

Рисунок 13 – выбор действия над таблицей

Для того чтобы сформировать и выполнить собственный SQL-запрос необходимо перейти в раздел SQL (рисунок 14), ввести SQL-запрос в соответствующее поле и нажать кнопку Пошел.

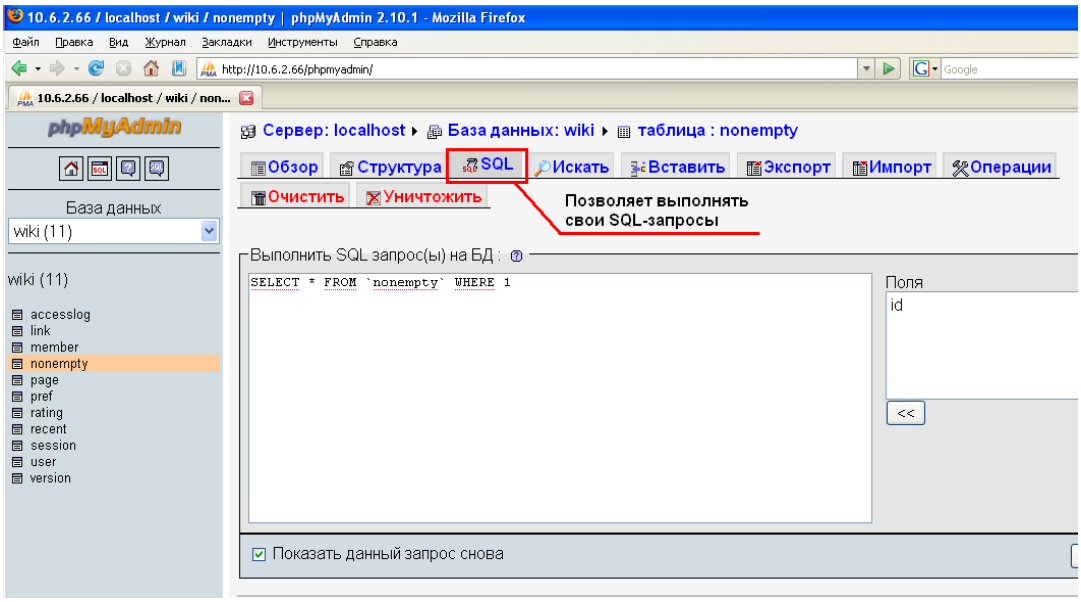

Рисунок 14 – выполнение собственного SQL-запроса

Приложение В

Некоторые команды Linux

ifconfig

Команда ifconfig используется для просмотра информации о настройках сетевых интерфейсов в Linux. Для того чтобы изменить настройки какого-либо интерфейса необходимо указать имя этого интерфейса в качестве первого аргумента команды ifconfig, а затем ввести новые параметры. Так для смены IP-адреса необходимо выполнить следующую команду:

```
ifconfig eth0 172.31.2.8
```

здесь eth0 – имя сетевого интерфейса, а 172.31.2.8 - новый IP-адрес данного интерфейса.

Использование команды ifconfig без указания аргументов приведет к выводу информации (текущих настроек) о всех интерфейсах.

ping

Команда ping чаще всего используется для проверки доступности удаленной рабочей станции. Для того чтобы узнать отвечает ли станция с IP-адресом 172.31.2.8 необходимо выполнить команду:

```
ping 172.31.2.8
```

mysql

Команда, которая позволяет подключиться к серверу баз данных MySQL и предоставляет возможность выполнения различных SQL-запросов.

man

Команда для вызова справки по всем Linux командам. Для контекстного поиска необходимо выполнить команду

```
man –k string
```

здесь *string* – искомая строка

mc

Команда mc предназначена для запуска файлового менеджер Midnight Commander, который позволяет искать, копировать, удалять, редактировать и производить другие действия над файлами.