ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра основ радиотехники и защиты информации

А.А. Антонов

ПРАКТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Учебно-методическое пособие
по проведению практических занятий

для студентов
специальности 10.02.05
очной формы обучения

Москва
ИД Академии Жуковского
2021

УДК 004.056
ББК 001.8
А72

Рецензент:
Петров В.И. – канд. техн. наук, доцент

Антонов А.А.
А72 Практические аспекты информационной безопасности [Текст] : учебно-методическое пособие по проведению практических занятий / А.А. Антонов. – М.: ИД Академии Жуковского, 2021. – 24 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Практические аспекты информационной безопасности» по учебному плану для студентов специальности 10.02.05 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 22.04.2021 г. и методического совета 22.04.2021 г.

УДК 004.056
ББК 001.8

В авторской редакции

Подписано в печать 28.05.2021 г.
Формат 60х84/16 Печ. л. 1,5 Усл. печ. л. 1,395
Заказ № 781/0519-УМП45 Тираж 40 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ

При подготовке к практическому занятию студенты должны:

уяснить цель и порядок проведения практического занятия;

изучить материалы, изложенные на лекциях и в рекомендуемой литературе.

На практическом занятии студент должен иметь конспект лекций и данное учебное пособие.

Практическое занятие начинается с опроса студентов по знанию теоретических положений практического занятия, а также проверки понимания решения типовых задач.

Далее студенты выполняют задания с последующим обсуждением полученных результатов.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. А.С. Масалков: Особенности киберпреступлений в России. Инструменты нападения и защита информации. М.: ДМК-Пресс, 2018.

2. А.А. Бирюков: Информационная безопасность: защита и нападение. М.:ДМК Пресс, 2012.

3. А.С. Панов: Реверсинг и защита программ от взлома. Спб.: БХВ-Петербург, 2006.

4. Ю.А. Семенов Обзор уязвимостей, некоторых видов атак и средств защиты (http://book.itep.ru/6/intrusion.htm).

Практическое занятие № 1

Безопасность веб-приложений. Ручной базовый анализ

1. Цель занятия - научиться анализировать и исследовать веб-приложения.

2. Краткие теоретические сведения

Одним из первых этапов анализ веб-приложения является базовый ручной анализ. Первым действием ручного анализа обычно является исследования исходного кода с помощью веб-браузера. Однако, не стоит думать, что будет отображён исходный код серверной части веб-приложения (то есть исходный код языка на котором написана серверная часть веб-приложения, например, PHP, Python и др.).

При использовании браузера можно получить только исходный код HTML, JS, CSS элементов веб-приложения, что чаще всего поможет лишь собрать некую информацию, касающуюся клиентской части веб-приложения.

Несмотря на это, иногда можно получить информацию об внутреннем устройстве веб-приложения или информацию касающуюся системных файлов.

а) Анализ исходного кода.

Для того чтобы открыть исходных код, необходимо в используемом браузере открыть выпадающее меню и выбрать пункт “View Page Source” , как представлено на рисунке 1.

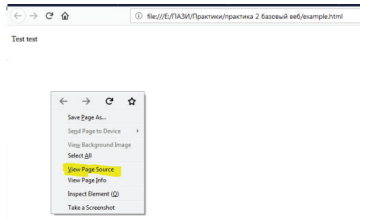

Рисунок 1 – Открытие страницы с исходным кодом

После этого в браузере отобразиться исходный код, в котором можно увидеть все HTML заголовки и тэги, а также комментарии, что часто бывает очень полезно, если разработчики оставили там какую-либо информацию, касательно работы веб-приложения, пример представлен на рисунке 2.

Рисунок 2 – результат открытия страницы с исходным кодом

Для отображения кода, также можно написать в браузере в строке запроса перед адресом сайта “view-source:”.

Часто в заголовке кода можно заметить секцию подключения сторонних файлов со стилями или JS-кодом для клиентской части. Это могут быть файлы как на данном сайте, так и файлы на удалённых сайтах, на которые ваш браузер должен будет обратиться, как представлено на рисунке 3.

Рисунок 3 – Секция подключения сторонних файлов

В данных файлах могут содержаться полезные данные, однако не стоит на них зацикливаться, особенно если это часто используемые стили или JS-компоненты.

б) Robots.txt

Данный файл может быть полезен при анализе большого веб-приложения, часто неопытные разработчики могут оставить там ссылки на файлы, которые хотят скрыть от поисковых роботов, таких как Google, Яндекс и других.

Для доступа к этому файлу достаточно обратиться к нему с корня сайта: http://ip-address/robots.txt, как представлено на рисунке 4.

Рисунок 4 – Использование файла Robots.txt

Далее целесообразно проверить каждую из ссылок и просмотреть что от нас хотели скрыть, хотя и довольно простым способом.

в) Исследование элементов. Сеть.

В меню, через которое открывает исходный код сайта, можно открыть «Инструменты разработчика» которые позволяют подробно изучить ответы от веб-приложения, элементы и их составляющие, как представлено на рисунке 5. Чаще всего исследователи пользуется ими для базового анализа ответов от веб-приложения, просмотра заголовков и прочей сетевой информации.

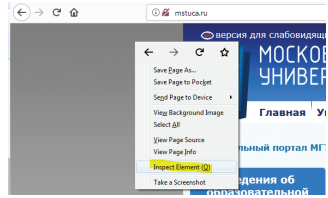

Рисунок 5 – Открытие инструментов разработчика

После этого внизу браузера откроется панель с несколькими вкладками, для открытия вкладки отвечающей за ответы от веб-приложения необходимо кликнуть по названию “Network”, как представлено на рисунке 6.

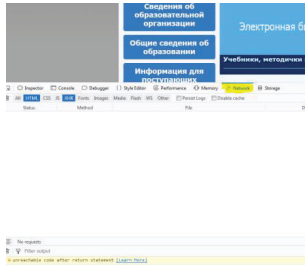

Рисунок 6 – Открытие инструментов разработчика

После этого нужно перезагрузить страницу и будут отображены запрос и ответ от веб-приложения, как представлено на рисунке 7.

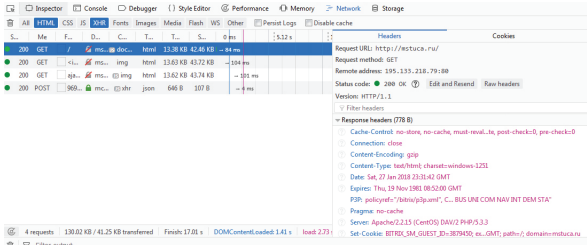

Рисунок 7 – Запрос и ответ веб-приложения

Слева можно увидеть заголовки запроса от нашего браузера (клиента) и заголовки ответа от веб-приложения (сервера), иногда в заголовках можно найти нестандартные поля, или, например, куки – остаточные файлы, в которых могут содержаться какие-либо данные. Также куки могут быть уязвимы, если обладают слабой криптографий, при этом храня важные данные, например, данные об уровне доступа аккаунта.

Куки можно увидеть в обоих типах заголовках, однако они будут находится в разных полях. В заголовках клиента это Cookie: xxxx, а в заголовках сервера это поле Set-Cookie: xxxx.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.
2. Загрузить операционную систему Kali linux с заранее подготовленной загрузочной флэш карты. Необходимо воспользоваться горячими клавишами «F12» и «Esc» для выбора загрузки операционной системы.
3. Используя командную строку браузера ввести адрес: 172.21.45.151/pazi, открыть практическое занятие № 1, где представлено теоретическое описание и задания.
4. Получить вариант задания у преподавателя.

5. Произвести базовый анализ 3 различных веб-приложений, найдя контрольные значения на каждом из веб-приложений. На одном веб-приложении может быть не одно контрольное значение, а несколько.

6. Составить отчёт.

4. Содержание отчета

1. Титульный лист.

2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).

3. Вывод. Стиль оформления отчёта – научно-технический.

Практическое занятие № 2

Заголовки сервера (HTTP заголовки)

1. Цель занятия - получение практических навыков работы с набором инструментов BurpSuite для тестирования безопасности веб-приложений и сайтов.

2. Краткие теоретические сведения

а) Заголовки сервера

Знание заголовков ответа от сервера может помочь с анализов веб-приложения, например, для понимания на какие заголовки нужно обращать больше внимания при анализе потенциальных уязвимостей.

На практическом занятии используется протокол прикладного уровня передачи данных HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста»), так же его расширением протокол HTTPS (S - Secure) — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности.

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

Стартовая строка (англ. Starting line) — определяет тип сообщения;

Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;

Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

В стартовой строке могут быть следующие различные типы сообщений.

В случае запроса:

- GET - запрашивает представление указанного ресурса.
- HEAD - запрашивает заголовки, идентичные тем, что возвращаются, если указанный ресурс будет запрошен с помощью HTTP-метода GET
- OPTIONS - используется для описания параметров соединения с целевым ресурсом.

- DELETE - удаляет указанный ресурс.
- PUT - создает новый ресурс или заменяет представление целевого ресурса, данными представленными в теле запроса.
- PATCH - применяет частичную модификацию к ресурсу.
- POST - предназначен для отправки данных на сервер. Тип тела запроса указывается в заголовке Content-Type.
- CONNECT - запускает двустороннюю связь с запрошенным ресурсом.
- TRACE - выполняет проверку обратной связи по пути к целевому ресурсу, предоставляя полезный механизм отладки.

  В случае ответа:
- 1xx - информирование о процессе передачи.
- 2xx - информирование о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса, сервер может ещё передать заголовки и тело сообщения.
- 3xx - сообщает клиенту, что для успешного выполнения операции необходимо сделать другой запрос.
- 4xx - указание ошибок со стороны клиента.
- 5xx - информирование о случаях неудачного выполнения операции по вине сервера

  На практическом занятии используются методы HTTP (типы запросов).

Метод HTTP - последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Обратите внимание, что название метода чувствительно к регистру.

Например, метод “GET” - Используется для запроса содержимого указанного ресурса.

Или “HEAD” - Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

Заголовки HTTP (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой. Один из самых важных заголовков - cookie.

Cookie — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.

Параметр User-Agent - показывает программное обеспечение клиента и его характеристики. По данному параметру веб-приложение может определить с помощью какого браузера было произведено обращение к серверу. Данный параметр не является обязательным при запросе, следовательно, его можно не использовать при запросе к веб-приложению, однако часто это может

распознаваться системами обнаружения вторжения, как сканирующие запросы, так как в сканерах обычно не используются заголовки User-Agent.

Заголовок Referer - содержит в себе URL после которого был сделан текущий запрос, то есть фактически определяет место сайта от куда вы были перенаправлены или сами перешли на текущую страницу.

В целом все заголовки разделяются на четыре основных группы:

General Headers (Общие заголовки) — используются в запросах и ответах.

Request Headers (Заголовки запроса) — используются только в запросах.

Response Headers (Заголовки ответа) — используются только в ответах.

Entity Headers (Заголовки сущности) — сопровождают каждую сущность сообщения.

Более подробно про все HTTP-заголовки можно почитать в сети Internet.

б) Набор инструментов утилиты BurpSuite.

Для удобной работы с заголовками обычно используют BurpSuite, который работает по принципу перехватывающего прокси сервера.

Данный инструмент покрывает почти всё, что может быть необходимо исследователю при анализе (имеется ввиду профессиональная версия данного инструмента).

Он является крайне функциональной платформой для анализа безопасности веб-приложений.

Основной режим работы BurpSuite представляет собой перехватывающий прокси для протоколов httpи https, соответственно для корректной работы BurpSuite, необходимо в вашем браузере настроить порт для прохождения трафика, далее утилита будет перехватывать и останавливать все запросы, идущие через ваш браузер, что позволит вам подменять их на лету, также вы можете отключить автоматическую остановку запросов и потом просмотреть их в истории запросов.

Запуск программы представлен на рисунке 8.

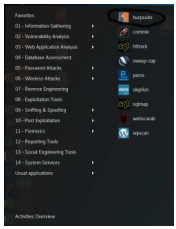

Рисунок 8 – Запуск программы BurpSuite

Burp Suite имеет графический интерфейс и запускается либо с левой панели быстрого доступа, либо через вкладку Applications->3 WebApplicationAnalysis. После запуска мы должны увидеть следующее окно, представленное на рисунке 9.

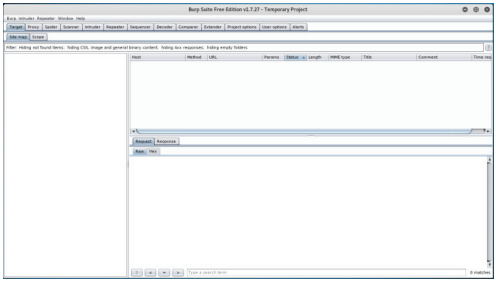

Рисунок 9 – Окно программы BurpSuite

Теперь необходимо настроить прокси, для этого нужно в верхнем меню перейти во вкладку Proxy->Options, далее нужно запустить прокси, это делается путём выставления флага Running, как представлено на рисунке 10.

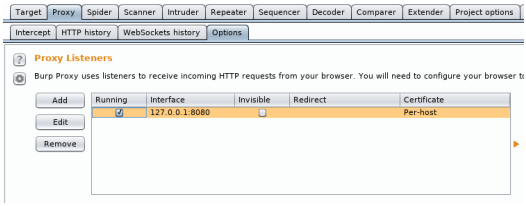

Рисунок 10 – Выставление флага в программе BurpSuite

Если у вас нет прокси сервера, то необходимо его добавить вручную, как представлено на рисунке 11.

В поле Bindtoport необходимо написать порт, на котором будет работать прокси, флаг Loopbackonly означает, что бинд сервера будет происходить на локальный хост.

После запуска утилиты BurpSuite и запуска прокси сервера необходимо настроить браузер на работу через наш прокси сервер.

Рисунок 11 – Добавление прокси-сервера в программе BurpSuite

Если вы используете всё на KaliLinux, то скорее всего вашим браузером для работы является Mozzila, для неё всё настраивается довольно просто,

заходим в настройки, далее расширенные настройки и там находим настройки сети, как на рисунке 12.

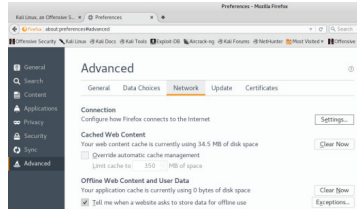

Рисунок 12 – Добавление прокси-сервера в программе BurpSuite

Далее нажимаете на настройки в выставляем наши параметры, как представлено на рисунке 13.

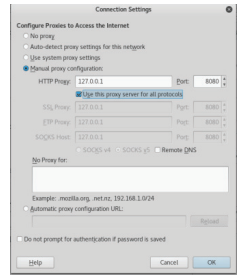

Рисунок 13 – Выставление параметров в программе BurpSuite

После применения настроек все запросы, которые идут через браузер будут проходить через наш перехватывающий прокси сервер и останавливаться, это мы можем увидеть в меню Proxy-> Intercepter в Burp Suite, как представлено на рисунке 14.

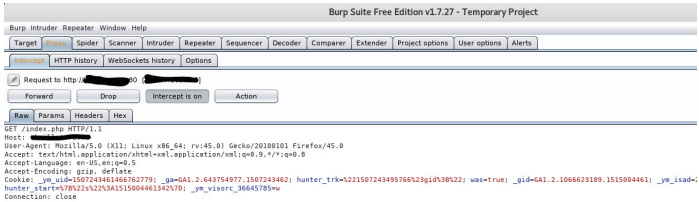

Рисунок 14 – Прохождение запросов, через прокси сервер

Если вы хотите подправить запрос вы можете изменить какие-либо данные в тексте запроса и нажать кнопку Forward что отправит запрос далее, если вы хотите не отправлять запрос можете нажать кнопку Drop.

Если у вас нет необходимости править запросы на лету, вы можете отключить опцию автоматического перехвата запросов. Для этого просто

нажмите на кнопку справа от Drop и увидите, как она превратиться в Interceptisoff.

Это значит, что перехват не будет осуществляться и вам не нужно будет каждый раз отвлекаться и вручную подтверждать запросы, однако вся история запросов будет сохранена во вкладке HTTP history, как представлено на рисунке 15.

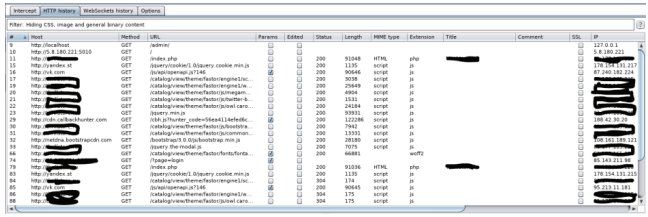

Рисунок 15 – История запросов во вкладке HTTP history

Нажав на любой запрос из списка внизу, вы увидите его подробное описание. Также вы можете отправить необходимый запрос в так называемый Repeater или повторитель, для того чтобы отправить его заново с измененными параметрами, это может быть удобно при тестировании различных инъекции или других атак.

Для этого нужно нажать на необходимый запрос правой кнопкой мыши и в выпадающем меню кликнуть на SendtoRepeater, как представлено на рисунке 16.

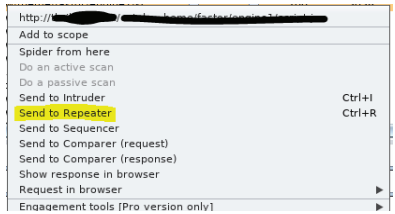

Рисунок 16 – Отправка запроса

После этого в верхнем меню перейти во вкладку Repeater. Там вы увидите запрос и сможете корректировать его и отправлять, наблюдая ответ в окне справа.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.

2. Загрузить операционную систему Kali linux с заранее подготовленной загрузочной флэш карты. Необходимо воспользоваться горячими клавишами «F12» и «Esc» для выбора загрузки операционной системы.

3. Используя командную строку браузера ввести адрес: 172.21.45.151/pazi, открыть практическое занятие № 2, где представлено теоретическое описание и задания.

4. Получить вариант задания у преподавателя.

5. Выполнить задания, используя утилиту BurpSuite, найдя в каждом задании контрольное значение.

6. Составить отчёт.

4. Содержание отчета

1. Титульный лист.

2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).

3. Вывод. Стиль оформления отчёта – научно-технический.

Практическое занятие № 3

Базовые уязвимости веб-приложений

1. Цель занятия - познакомиться с базовыми уязвимостями веб-приложений и с их использованием.

2. Краткие теоретические сведения

а) SQLi

Для представления механизма работы атаки SQLi, нужно представлять, что такое база данных и скрипты, в которых производятся запросы к базе данных, а также знать хотя бы на примитивном уровне SQL и иметь навык написания базовых запросов.

Обычная, в нашем понимании, БД состоит из множества таблиц. У каждой таблицы, естественно, есть столбцы и есть строки. Собственно, это ключевой момент.

Возьмем к примеру таблицу пользователей какого-либо интернет-магазина. Для каждого пользователя должно быть описано несколько параметров (логин, адрес электронной почты, дата регистрации, сохранённая карточка и т.д.). В итоге каждый столбец определяет какой-либо параметр пользователя, а каждая строка - конкретного пользователя. А в пересечении нужного нам столбца и строки будет информация о параметре нужного пользователя.

Для полного понимания можно привести пример:

Представим, что вы (скрипт) идёте в магазин (базу данных), и просите в магазине (создаёте SQL запрос и отправляете его в базу данных) продавца: “Мне нужна одна бутылка воды за 25 рублей”.

Теперь переделаем это в SQL запрос к базе данных, получим следующее: SELECT * FROM магазин WHERE (тип=’вода’ AND цена=25) LIMIT 1.

Собственно, в ответ на вашу просьбу (SQL запрос) вы (скрипт) получаете бутылку (информацию), и продавца (Базу данных) уже не волнует, что вы будете с ней делать, так как свою работу он выполнил. Вы можете ее выпить, вылить, подарить (обработать, вывести, рассчитать) и т.д.

Для полного понимания запроса в базу данных необходимо знать на базовом уровне язык SQL, однако даже без особых знаний можно заметить, что запрос выглядит понятным, так как используются довольно известные слова и они трактуются в данном случае однозначно.

Вопросы может вызвать разве что окончание запроса: “LIMIT 1” – данная конструкция объявляет, что необходимо выдать только первый найденный результат, если вдруг их больше одного (бутылок по такой цене может быть очень много, но нам нужна только одна).

Все остальные мнемоники (команды) запроса предельно понятны:

SELECT – выбери (возьми)

FROM – из (указывает от куда нужно взять, чаще всего после данной команды следует название одной из таблиц в текущей базе данных)

WHERE – где (указывает какие значения различных параметров должны быть у той строки которую мы хотим получить, по сути указывает критерии для того товара который мы хотим получить)

Обобщенно атака типа SQL инъекция (SQL injection) возникает в случае если злоумышленник может каким-то образом модифицировать запрос к базе данных. Рассмотрим всё тот-же пример с магазином.

Пусть вы решили пойти в магазин за кефиром, и специально написали на бумажке список продуктов, которые хотите купить, чтобы ничего не забыть. Предположим, что у вас дома оказался друг, который очень любит шоколад, однако ему нельзя есть его много и на его просьбу, вы ответили отказом и не стали записывать в листочек его пожелание.

Итого вы получаете листочек, в котором на каждой новой строке записаны продукты, которые вам необходимы в магазине.

Допустим, что каждый из продуктов будет представлять собой некий запрос к базе данных (магазину) как и в прошлом примере. Вы просто отдаёте продавцу свой листочек (скрипт передаёт запросы в базу данных) и он, читая очередную строку кладёт в ваш пакет тот или иной продукт (сохраняет результаты, того что удалось получить из базы данных).

Теперь представим, что в данный листок ваш друг всё-таки смог вписать шоколад (провел атаку SQL injection) и продавец, читая очередную строчку может увидеть следующее: “Масло за 100 рублей или шоколад за 150 рублей”.

В магазине может не оказаться масла, или продавцу долго за ним идти, и он положит вам в пакет шоколад. Продавец в данном случае не имеет представления о том, что вашему другу нельзя шоколад и вам он тоже особо не нужен, он просто выполняет свою работу, предоставляет продукты согласно переданному ему списку (база данных будет осуществлять работу по возврату

данных на основание входящих запросов, если они были составлены правильно).

Переведя это в SQL запрос получим следующее:

SELECT * FROM магазин WHERE (тип=’масло’ AND цена=’100’) OR (тип=‘шоколад’ AND цена=‘150’) LIMIT 1

Атака удалась потому что вы не проверили листочек перед тем как отдать его продавцу (скрипт не проверил входные данные перед формированием запроса и его отправки в базу данных).

SQLi может возникать в местах, где есть какие-либо входящие параметры, например, формы поиска товаров в интернет магазине или форма авторизации, которые есть на большей части всех веб-приложений. Любой не фильтруемый пользовательский ввод – почти всегда приводит к плачевным последствиям.

В случае с SQLi есть некоторые особенности, связанные с запросом, в который подставляются введённые пользователем данные. Например, тип значение, в которое поступают данные от пользователя может быть целочисленным или строковым, или стоять в нескольких скобочных вложениях, что немного затруднит написание зловредного запроса.

В случае с запросом на авторизацию, необходимо понимать, что как таковые данные с помощью такого запроса получить не удастся, так как такого рода запросы чаще всего рассчитываются на получение хэша от пользовательского пароля и сравнение его с высчитанным хэшом от входного пароля.

Большинство из SQLi обнаруживаются путём нарушения общего мнения о том, что должно быть введено в данное поле конечным пользователем, чаще всего срабатывает символ одинарной кавычки, так как многие из параметров – строковые.

Подставив в какой-либо параметр кавычку и получив ошибку, содержащую слова “SQL syntax error” или что-то подобное, можно сразу сказать, что скорей всего здесь есть SQLi.

Однако, есть варианты что кавычка фильтруется, или отключен отчёт об ошибках, в этом случае нужно прибегать к другим методам.

б) Local File Include (LFI) — уязвимость, которая позволяет удаленному пользователю получить доступ к нужной информации с помощью специально сформированного запроса к произвольным файлам.

LFI представляет собой подключение любого файла на сервере к вызываемому файлу. Что делит LFI на две ветки: выполнение содержимого подключаемого файла и чтение содержимого подключаемого файла. Уязвимости класса LFI чаще всего встречается в различных менеджерах файлов.

в) RCE (англ. Remote Code Execution) – компьютерная уязвимость, при которой происходит удаленное выполнение кода на взламываемом компьютере, сервере и т.п.

RCE - это гарантированный способ взлома сайтов и веб приложений. RCE - является одной из самых опасных уязвимостей.

Возможность удаленного внедрения кода в серверный скрипт в 100% случаев приводит к взлому ресурса. С помощью RCE злоумышленник сразу получает доступ к серверу атакуемого сайта, размещая на нем веб-шеллы, или любой другой вредоносный код.

Возможность использования RCE-уязвимости возникает из-за грубейших ошибок разработки сайта, отсутствия фильтрации передающих параметров, использование небезопасных функций и приемов программирования.

Задание на практику:

Решить 3 предоставленных задания, найдя в каждом контрольное значение и составить отчёт.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.
2. Загрузить операционную систему Kali с заранее подготовленной флэш карты. Необходимо воспользоваться горячими клавишами «F12» и «Esc» для загрузки операционной системы linux.
3. Используя командную строку браузера ввести адрес:172.21.45.151/pazi, открыть практическое № 3, где представлено три задания.
4. Выполнить три задания, сформулировать выводы, рекомендации для устранения уязвимостей.
5. Составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).
3. Вывод. Стиль оформления отчёта – научно-технический.

Практическое занятие № 4

Обратная разработка. Базовый анализ исполняемых файлов

1. Цель занятия - получение базовых навыков анализа исполняемых файлов и использования инструментов для обратной разработки.

2. Краткие теоретические сведения

Для выполнения практического занятия целесообразно использовать различные инструменты, однако все задания можно выполнить с помощью IDA Pro.

1. Первоначальные действия.

При анализе любых исполняемых файлов чаще всего в начале стоит ряд задач, решаемых довольно быстро. Например, определение микропроцессорной архитектуры команд или разрядности файла. Чаще всего после решения данных

задач и загрузки исполняемого файла в IDA Pro, производится первичный анализ файла. В первую очередь просматриваются строки, имеющиеся в исполняемом файле. Строки могут помочь понять некоторую логику работы приложения, так-как можно проследить, где какая строка используется, в связи с этим понять назначение того или иного блока. Список строк в IDA Pro можно получить с помощью нажатия “Shift+f12”.

В открывшемся окне будут отображены все автоматически распознанные строки, для того, чтобы попасть на само место хранения строки в исполняемом файле достаточно 2 раза нажать левую клавиши мыши по интересующей строке.

Для того чтобы узнать, где данная строка используется, можно получить все перекрёстные ссылки на неё путём нажатия горячей клавиши “x”, при выделенной метке, отвечающей за строку. Пример представлен на рисунке 17.

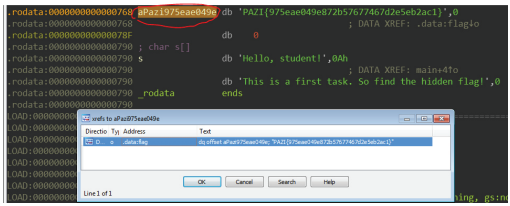

Рисунок 17 – Фрагмент окна дизассемблера IDA Pro

Необходимо навести на метку (имя) для данной строки (обычно располагается слева от самого значение) и нажать горячую клавишу “x”, после откроется окно (как показано на рисунке), где будут отображены все ссылки (то есть все использования данной строки, если они имеются) на данную строку в программе.

Исходя из этих данных, можно определять предназначение тех или иных блоков (например, блок, отвечающий за верно введённый ключ).

2. Представление различных типов данных.

IDA Pro умеет автоматически определять различные типы данных, однако зачастую она делает это не совсем верно, и требуется «помогать» ей в этом. Для этого существуют функции преобразования данных. Например, для преобразования какого-либо значения в ASCII символ можно использовать клавишу “R”. Пример представлен на рисунке 18.

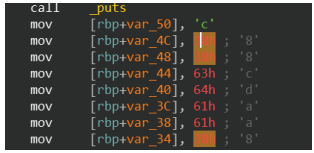

Рисунок 18 – Преобразование значения в ASCII символ

После нажатия на “R”, выделенное нами значение (там, где находится курсор) преобразуется в ASCII символ ‘8’, соответственно таким же образом можно преобразовать все последующие значения.

3. Анализ простых условий.

Практически все программы имеют многочисленные ветвления. Это связано с наличием различны сравнений, проверки условий и прочего. Для анализа простых условий можно использовать графическое представление, для более удобного отображения ветвления. Пример представлен на рисунке 19.

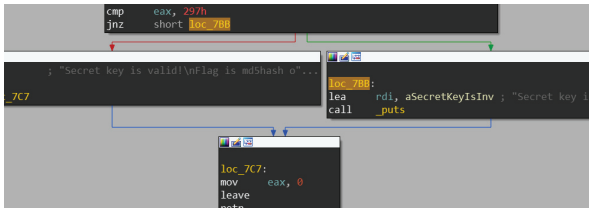

Рисунок 19 – Графическое представление дизассемблера

Можно заметить мнемонику “cmp” перед мнемоникой “jnz”, которая является условным переходом в один из блоков, если переход не произойдёт, продолжится выполнение следующей команды, которая находится в соседнем блоке. Как можно заметить, необходимо чтобы в регистре eax, к моменту сравнения находилось значение 297h (h-указание на то, что величина является шестнадцатеричной).

Для решения данной задачи необходимо проследить, какие изменения происходят с данным регистром до этого момента и попытаться выяснить, от каких данных и действий зависит конечное значение. Для этого необходимо проанализировать небольшой участок кода, представленный на рисунке 20.

```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 0Fh
lea     rdi, format     ; "Enter the secret key: "
mov     eax, 0
call    _printf
lea     rax, [rbp+var_8]
mov     rsi, rax
lea     rdi, aD         ; "%d"
mov     eax, 0
call    __isoc99_scanf
mov     eax, [rbp+var_8]
add     eax, 4
sar     eax, 1
xor     eax, 6
xor     eax, [rbp+var_4]
cmp     eax, 297h
jnz     short loc_7BB
```

Рисунок 20 – Анализ фрагмента файла дизассемблером

Как можно заметить, в регистр eax помещается значение локальной переменной, которая инициализируется путём ввода с клавиатуры. То есть изначально в регистре eax будет находится введённое пользователем значение.

Далее это значение проходит через ряд простым математических операций, таких как сложение, побитовый сдвиг вправо, сложение по модулю два с числом, и сложение по модулю два с другой локальной переменной, ранее проинициализированной.

Необходимо провести данные операции в обратную сторону с финальным числом (то есть с числом, с которым производится сравнение) и найти число, которое необходимо ввести с клавиатуры для получения верного выполнения финальной проверки.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.
2. Получить задания у преподавателя.
3. Произвести обратную разработку 3 исполняемых файлов, в первом и втором исполняемых файлах найти контрольные значения.

В третьем исполняемом файле найти число (или числа, если их несколько), при вводе которого (которых) происходит выдача сообщения о верности секретного ключа.

4. Составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).

Решения всех заданий должны быть подробно описаны и содержать скриншоты. Для последнего задания должна быть приведена формула, по которой происходит преобразование введённого значения, и обоснована верность полученного ответа

3. Вывод. Стиль оформления отчёта – научно-технический.

Практическое занятие № 5

Обратная разработка. Анализ алгоритмов защиты программного обеспечения от копирования

1. Цель занятия - получение базовых навыков продвинутых навыков анализа исполняемых файлов и использования инструментов для обратной разработки.

2. Краткие теоретические сведения

Для практической реализации необходимо использовать дизассемблер IDA. Помимо самого процесса дизассемблирования, IDA прилагает усилия не

только для того, чтобы отличить байты данных от байтов кода, но и для того, чтобы точно определить, какой тип данных представлен этими байтами данных.

Для выполнения заданий, практического занятия необходимо детально изучить следующие пункты.

а) Сбор общей информации о файле.

При получении исполняемого файла, неизвестно его предназначение, кроме того, что он может выполниться и производить какие-либо действия. Для того чтобы узнать, какие точно действия он производит и каким образом, потребуется провести исследование, которое начинается с общего анализа файла.

Первое что необходимо узнать это платформа для его исполнения и разрядность файла. Данную информацию можно получить при анализе файла в шестнадцатеричном редакторе. При анализе заголовка файла, очень можно понять платформу, для которой предназначен данный двоичный файл, также в заголовочной секции можно найти признаки принадлежности к той или иной разрядности.

После анализа полученной информации, касающейся строк, можно переходить непосредственно к восстановлению логики работы приложения.

б) Статический анализ дизассемблированного кода

Существуют различные подходы к решению данного вопроса, при анализе крупных проектов не принято восстанавливать логику работы всей системы, обычно исследуются конечные части, выполняющие несколько функций. При таком анализе обратная разработка начинается с какой-либо конечной функции, действие которой вы однозначно можете определить.

Однако в случае данной лабораторной работы можно начинать обратную разработку прямо с функции main(), так как функционал приложения крайне мал и однообразен, поэтому запутаться в функциях невозможно.

Для удобного исследования кода рекомендуется применять различные типы отображения в IDA. Типы отображения кода можно менять с помощью клавиши пробел.

Режим графового отображения кода позволяет быстро ориентироваться между логическими блоками и быстрее анализировать логические условия,

в) Преобразование алгоритмов с языка низкого уровня на язык программирования высокого уровня.

После анализа алгоритмов, реализованных в исполняемом файле, часто возникает потребность реализовать их на языке программирования высокого уровня, реализация может быть абсолютно идентичная по функциональности, но иметь другие параметры или носить иной смысл.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.
2. Получить задания у преподавателя.

3. Произвести обратную разработку двух исполняемых файлов, в обоих файлах ввести верный серийный ключ, который запрашивает программа.

4. Составить отчёт.

4. Содержание отчета

1. Титульный лист.

2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).

Решения всех заданий должны быть подробно описаны и содержать скриншоты.

3. Вывод. Стиль оформления отчёта – научно-технический.

Практическое занятие № 6

Обратная разработка приложений. Бинарные уязвимости

1. Цель занятия - получение практических навыков эксплуатации простейших бинарных уязвимостей.

2. Краткие теоретические сведения

а) Общие понятия о бинарных уязвимостях.

Бинарные уязвимости являются одним из самых сложных классов уязвимостей в современном мире практической информационной безопасности связанной с анализом кода.

Существует два больших класса уязвимостей: кода и архитектуры. Уязвимости кода непосредственно связаны с неправильными обработками данных или недостаточной проверкой получаемых значений. Уязвимости архитектуры представляют собой реализацию каких-либо возможностей системы вне задуманных целей для это возможности.

б) Переполнение буфера.

Данная уязвимость считается самой популярной при исследованиях, несмотря на то, что разработано уже множество методик, позволяющих компилятору защищаться от подобного рода уязвимостей, они всё равно обнаруживаются и эксплуатируются, особенно в старом программном обеспечение, которое не редко входит в состав нового программного обеспечения как некие полезные модули.

Суть данной уязвимости заключается в том, что размер входного буфера данных зачастую не проверяется при перемещение его из одного места в другое или просто при считывании. При этом данная уязвимость не обязательно должна быть связана с переполнением стека или кучи с возможностью выполнения кода, также можно и перезаписывать некоторые локальные переменные при возникновении переполнений буфера.

Рассмотрим на примере, как представлено на рисунке 21.

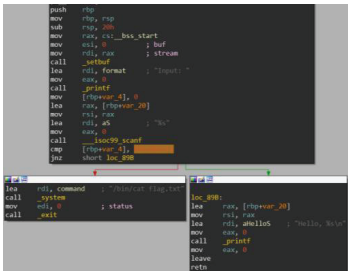

Рисунок 21 - Пример уязвимости переполнения буфера.

Представлен пример переполнения буфера, при этом в данном случае при переполнение будет произведена перезапись локальной переменной, но если попытаться перезаписывать значения дальше вплоть до адреса возврата, то произойдёт ошибка, связанная с нарушением целостности стека. Эта ошибка связана с наличием защищающих механизмов в компиляторе. Одним из таких механизмов является так называемая «стековая канарейка», которая находится перед адресом возврата и соответственно нельзя перезаписать адрес возврата не перезаписав её, а её перезапись влечёт за собой аварийное завершение работы программы, так как обработчик обнаружит нарушение целостности стека.

Перезапись переменной произойдёт потому, что в функцию scanf() не был явно передан аргумент отвечающий за количество считываемых символов, однако массив в который данные символы будут считываться ограничен 16 байтами, которые располагаются на стеке. В данном примере это показано довольно искусственно, т.к. по условиям видно, что для того чтобы выполнилась команда чтения флага одна из локальных переменных должна быть равна значению **0xdeadbeef**, однако данная переменная нигде не изменяет своё значение из чего можно сделать вывод что от нас требуется изменить это значение при помощи переполнения буфера.

Для того, чтобы высчитать верное количество, байт которое нужно передать, чтобы попасть на перезапись необходимой переменной можно воспользоваться IDA и посмотреть разбиение стека для данной функции.

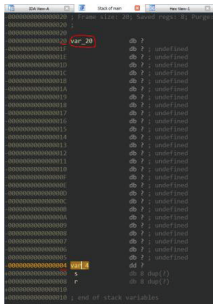

Рисунок 22 - Разбиение стека функции main.

Данную операцию можно выполнить с помощью горячих клавиш “Ctrl+K”, после их нажатия откроется новая вкладка, в которой будет указано разбиение стека для данной функции.

Как можно заметить по рисунку 22, от начала нашего буфера, который хранит введённые значения до адреса необходимой переменной 0x20-0x4 = 0x16 – байт, в переводе на десятичную систему 32-4=28 байт. То есть после передачи 28 байт следующие байты будут перезаписывать данную переменную.

Данный вывод довольно просто проверяется при помощи запуска исполняемого файла с передачей в него необходимых параметров, как представлено на рисунке 23.

```
root@kali:~/pazi_pwn/task1# python -c 'print "a"*26 + "\xef\xbe\xad\xde"' | ./a.
out
Input: Hello, aaaaaaaaaaaaaaaaaaaaaaaa
root@kali:~/pazi_pwn/task1# python -c 'print "a"*28 + "\xef\xbe\xad\xde"' | ./a.
out
Input: PAZI{FLAG}
root@kali:~/pazi_pwn/task1# cat flag.txt
PAZI{FLAG}
root@kali:~/pazi_pwn/task1#
```

Рисунок 23 - Эксплуатация уязвимости.

Данный пример довольно искусственен, однако хорошо демонстрирует базовый принцип уязвимости переполнения буфера.

в) Уязвимость форматной строки.

Суть данной уязвимости заключается в том, что введённые пользователем данные могут передаваться в функции вывода на экран, при этом формат данных не будет указан явно, что позволяет пользователю выводит те данные на экран которые ему необходимы, например, содержание стека функции. В контексте атаки данная уязвимость может быть полезна, например, для получения как раз той самой «стековой канарейки», при её получении мы можем перезаписать её верным значением и после перезаписать необходимым нам значением адрес возврата.

Рассмотрим пример. На рисунке 24 видно, что введённые пользователем данные в программ считываются и добавляются в буфер, далее данный буфере предаётся в функцию printf() однако, формат передаваемых данных никак не указывается, что позволяет манипулировать выводом данной функции в различных целях.

Целью данного примера является возможность чтения контрольного значения PAZI{flag}, которое можно заметить в функции. Однако и на практике можно с пользой применять данную уязвимость, например, если она была найдена во встраиваемой системе под которую нет отладчика, с помощью данной уязвимости можно следить за значениями на стеке.

Для выполнения цели предусмотренной данным искусственным примером достаточно произвести определённый форматный ввод и немного обработать вывод. Ввод представлен на рисунке 25, после получения объёма

шестнадцатеричных цифр можно выделить в них ASCII-коды формата флага и найти флаг.

Рисунок 24 - Пример уязвимости форматной строки.

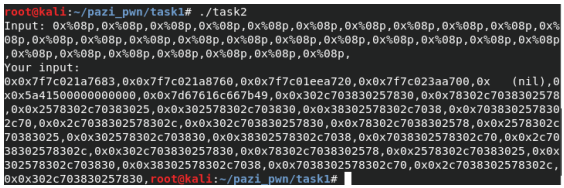

Рисунок 25 - Пример эксплуатации уязвимости форматной строки.

3. Порядок выполнения заданий.

1. Изучить теоретические сведения.
2. Получить задания у преподавателя.
3. Необходимо найти три уязвимости, по одной в каждом бинарном файле. Для первого файла эксплуатация уязвимости заключается в прочтении файла flag.txt Для второго файла эксплуатация заключается в выводе флага на экран. Для третьего файла эксплуатация заключается в выводе сообщения "Correct!" на экран. Демонстрации эксплуатации уязвимостей производятся путём запуска бинарных файлов под операционной системой Linux и подаче на вход таких данных, которые продемонстрируют уязвимость и выведут верный ответ на экран.
4. Составить отчёт.

4. Содержание отчета

1. Титульный лист.
2. Описание по каждому пункту проведённых вами действий с изображениями (скриншотами).
3. Вывод. Стиль оформления отчёта – научно-технический.